ARCH 1987

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

THE BANDWIDTH BOTTLENECK: Compressing Image Data

Squeezing Text Files

Optimizing Integer Multiplications

Webster's vs. K&R

80386 Resources

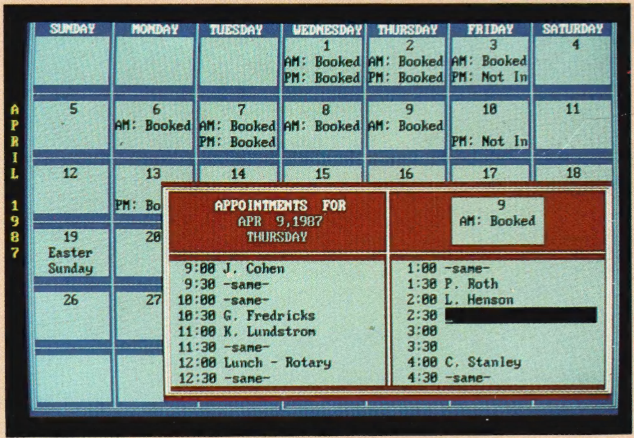
Languages:

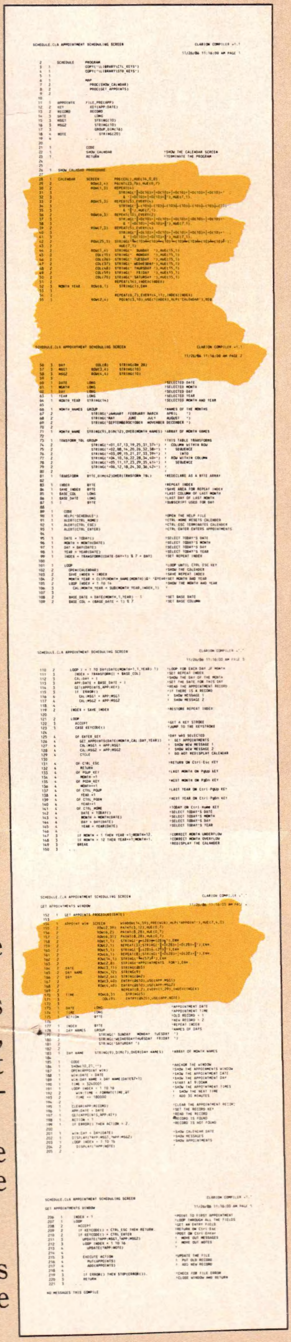
C Text Formatter
Object-Oriented LISP
BASIC Modules and Libraries
Assembly vs. High-Level Languages

03

0 38351 16562 8

BOB STANTON HAD A GREAT IDEA. AN HOUR LATER HE WAS TESTING IT.





Appointments. Everybody takes them — dentists, autobody shops, dance instructors. And lots of computer applications need appointment screens.

Bob thought that a calendar made a terrific graphic metaphor for taking appointments. Simply use the arrow keys to pick an open date, then press the Enter key, and up pops an appointment window.

Lucky for Bob, he's a CLARION programmer, one of a fast growing cadre of super-productive application developers.

With CLARION's Screener utility, he painted a white calendar on a black background. Then he drew a white-on-blue track around the page and between the days. He typed in the days of the week — and *voila!* — a calendar!

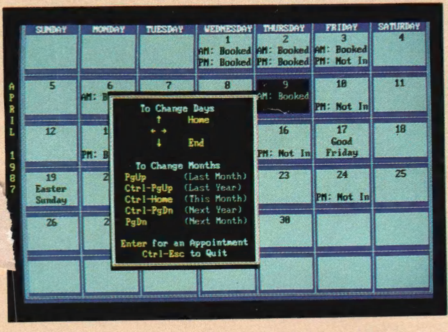
CLARION knows that a PC monitor is refreshed from memory, so it treats a screen layout like a group of variables. Just move data to a screen variable, and it shows up on the monitor.

Bob set up dimensioned screen variables for the days of the month and a screen pointer for selecting a date, and he was done. Then Screener generated the code.

Then Bob drew the appointments window, built an appointment file, filled in the connecting code and tested it — ONE HOUR AFTER HE STARTED!

Testing was a breeze. Screener doesn't just write code, it compiles your source, displays a screen, gets the changes, then replaces the old code in your program.

So here are Bob's appointment screens. You can see the source listing to the right. We marked all the code Screener wrote for him.



WHY CLARION?

Why are application developers everywhere changing to CLARION?

Because CLARION gives you all the tools you need: a coupled compiler and editor; screen, report, and help generators; an import/export utility; a sort/backup/restore utility; a formatted file dump; a DOS shell — and much more.

Because with CLARION's comprehensive data management routines, records can be locked and files shared on Novell®, 3COM®, IBM® PC Net & Token Ring, Multi Link®, and most other networks.

Because CLARION is *not* hardware locked or copy protected. Run-time systems are *free* and soon you will be able to translate CLARION into native machine code (.EXEs).

And best yet, the price of CLARION v1.1 is just $395 plus shipping and handling.

You'll need an IBM PC or true compatible with 320KB of memory and a hard disk drive. CLARION v1.1 also comes with a 30-day money back guarantee.

So call now and order CLARION v1.1. or ask for our detailed 16-page color brochure and reprints of major reviews.

CLARION®
from BARRINGTON SYSTEMS, INC.

800/354-5444

150 EAST SAMPLE ROAD **POMPANO BEACH, FLORIDA 33064** **305/785-4555**

Circle **no. 115** on reader service card.

TRANSFORM YOUR PC INTO A LISP MACHINE.

Not long ago, a specialized LISP machine was your only choice for serious AI development and delivery.

But now you can get LISP machine performance out of that ordinary IBM PC* sitting on your desk. With the **Gold Hill 386 LISP System.**

You simply plug in the System's HummingBoard™—unique 386-based hardware designed specifically for the LISP environment. Then you add the System's Golden Common LISP 386 Developer software.

That's all you need to transform your PC into a LISP machine. You can develop and deliver AI applications on your PC. And create your own expert systems. You'll get the kind of LISP performance you thought was only possible in a system costing ten times as much.

And if you don't need the whole system, Gold Hill can still offer you AI solutions. You can get GCLISP 286 Developer software for your PC.* And GCLISP 386 Developer software will be available for leading manufacturers' 386-based PCs.

Tomorrow's AI development tools for PCs are available today from Gold Hill. We'll prove it to you—just call to get the latest Gabriel Performance Benchmarks. You'll be amazed to learn what your PC is capable of. Call toll-free:

Gold Hill. The expert in AI on PCs.

1-800-242-LISP

In Mass.: (617) 492-2071
Gold Hill Computers, Inc.
163 Harvard St.,
Cambridge, MA 02139

GOLD HILL™

*The Gold Hill 386 LISP System requires an IBM PC XT, AT or compatible. GCLISP 286 Developer Software requires an IBM PC AT or compatible.

Circle **no. 291** on reader service card.

NEW! FROM BLAISE COMPUTING

C Tools Plus™

For The Programmer Whose Alphabet Begins & Ends With "C"

Today's programmers need more than yesterday's tools. Requirements such as removable windows and "sidekickable" pop-up utilities are changing the face of program design. You need to filter interrupts so that other resident programs still work. You need the ability to switch between multiple display pages and monitors. Today's technical demands are almost endless, but C TOOLS PLUS gives you what you need.

SOLID LIBRARY SUPPORT

Blaise Computing offers you solid library support that can meet all your demands and more. C TOOLS PLUS embodies the full spectrum of general-purpose utility functions that are critical to today's applications.

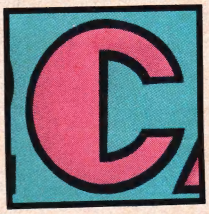
Here's just part of the PLUS in C TOOLS PLUS:

- ◆ **C TOOLS and C TOOLS 2** compatibility—two packages that receive rave reviews for quality, organization, usability and documentation.
- ◆ **FULL SOURCE CODE**
- ◆ **WINDOWS** that are stackable, removable, that support word wrap and that can accept user input.
- ◆ **INTERRUPT SERVICE ROUTINE** support for truly flexible, robust and polite resident applications.
- ◆ **MULTIPLE** monitor and display support, including EGA 43-line mode.
- ◆ **FAST DIRECT VIDEO ACCESS** for efficiency that will not constrain good program design.
- ◆ **DOCUMENTATION, TECHNICAL SUPPORT** and attention to detail that have distinguished Blaise Computing products over the years.

*C TOOLS PLUS supports the Microsoft (and IBM) 3.00 and Lattice 3.00 C compilers and is just **$175.00.***

Also Available Are:

***C VIEW MANAGER**—A kit for building data entry screens and menus. Begin by designing on-screen what the operator will see; call upon our library functions from your program to display the screens and retrieve the data. Just $275, including all library source code.*



***C ASYNCH MANAGER**—provides the crucial core of hardware interrupt support needed to build applications that communicate. It also includes the "XMODEM" file-transfer protocol and support for Hayes-compatible modems. All source code is included for $175.*

***C TOOLS & C TOOLS 2**—an indispensable combination still available at a low price of $175, including all source code. See review in PC Tech Journal, 6/85.*

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

ORDER TOLL-FREE 800-227-8087!

YES, send me the PLUS I need! Enclosed is $______ for C TOOLS PLUS. (CA residents add 6½% Sales Tax. All domestic orders add $10.00 for Federal Express shipping.)

Name: ______ Phone: (___) ______

Shipping Address: ______

City: ______ State: ____ Zip: ______

VISA or MC #: ______ Exp. Date: ______

Circle **no. 159** on reader service card.

About the Cover
It's easy to reduce the size of a file so you can transmit it more quickly. The trick is to do it in such a way that you can recreate the original file on the other end.

This Issue
Given a fixed-capacity channel and a quantity of information to transfer, how do you move more information down the channel per unit of time than the channel will support? That apparently impossible challenge is the bandwidth problem, a topic that we intend to address in various ways throughout 1987. The expanded, extended, or otherwise enlarged memory space of today's microcomputers, in conjunction with the processing power of CPUs such as the 80386 and 68020, only makes the information bottlenecks in computer systems all the more apparent. Two solutions to the bandwidth dilemma lead off this issue.

Next Issue
In the area of artificial intelligence, expert systems are now old hat. The next area of growth in AI must be in techniques that allow the program to acquire new information—to learn. Our annual AI issue will include an implementation of a classic expert system, but it will also look forward with an example of software that mirrors the structure of the brain and learns by experience, much as clusters of neurons may learn.

bandwidth topic
entry point

IS GETTING THE ANSWER TO SOFTWARE PROBLEMS A BIGGER PROBLEM THAN THE PROBLEM?

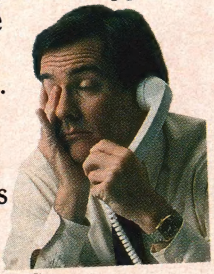
Don't stay on hold when there's help online from CompuServe® Software Forums.

The new upgraded version of your software locks up. And every time you reboot, you get stuck in the same place in the program.

You've chucked the manual, because you've done exactly what it tells you to do six times already. So you call the software company.

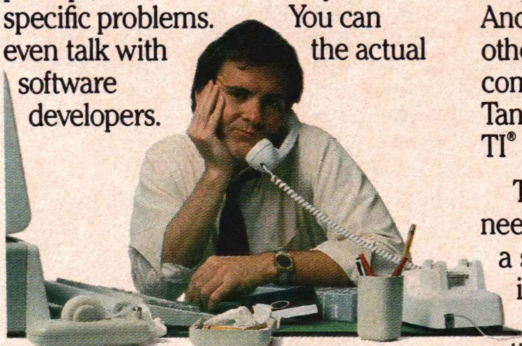
Now you spend half a day beating your head against a brick wall of busy signals, ranting at recorded messages, hanging around on hold. And you still don't get the solution to your problem.

Meanwhile, progress is stopped and your profits are dribbling away. But wait. There's help...

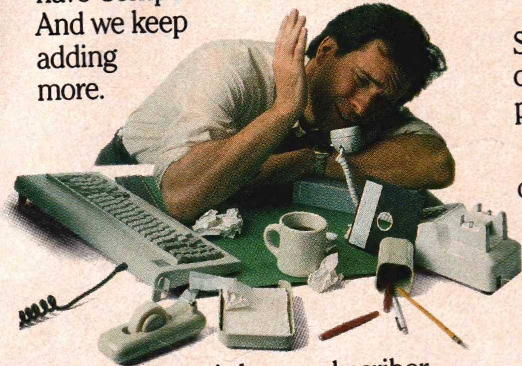


Several prominent, progressive software publishers recognize this problem, and working with CompuServe, have developed a solution—CompuServe Software Forums.

Now you can go online with experts from the companies that produced your software and get prompt, written answers to your specific problems. You can even talk with the actual software developers.



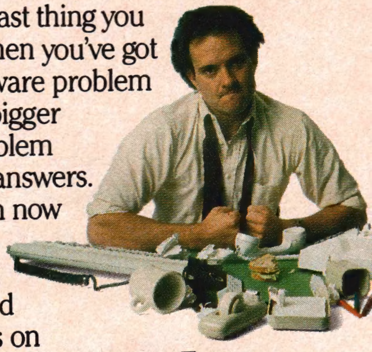
Adobe Systems®, Aldus®, Ashton-Tate®, Autodesk®, Borland International®, Creative Solutions®, Digital Research®, Living Videotext®, Lotus® Inc., Microsoft®, MicroPro®, Misosys Inc.® and Software Publishing® all have CompuServe Software Forums. And we keep adding more.



CompuServe's large subscriber base also puts you in touch with thousands of other, often more experienced, users of the same software. You'll find they can give you lots of creative ways to get the most out of your software.

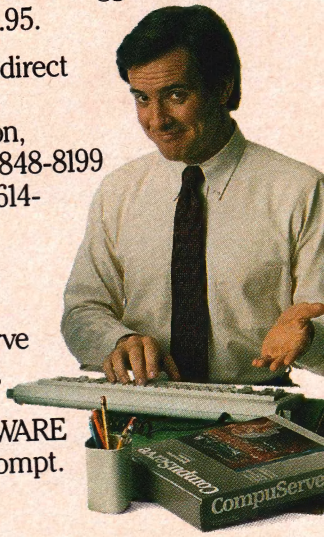
And software forums are the best way to learn about product updates, new product announcements, new ways to expand the uses of your software, and offer free uploads of your own programs.

Our online electronic magazines frequently publish software reviews. And you can find help for many other software products in our other computer-related forums for IBM®, Tandy®, Atari®, Apple®, Commodore®, TI® and others.



The last thing you need when you've got a software problem is a bigger problem getting answers. So, from now on, get prompt, informed answers on CompuServe Software Forums.

To buy your CompuServe Subscription Kit, see your nearest computer dealer. Suggested retail price is $39.95.



To order direct or for more information, call 800-848-8199 (in Ohio, 614-457-0802).

If you're already a CompuServe subscriber, just type GO SOFTWARE at any ! prompt.

CompuServe®

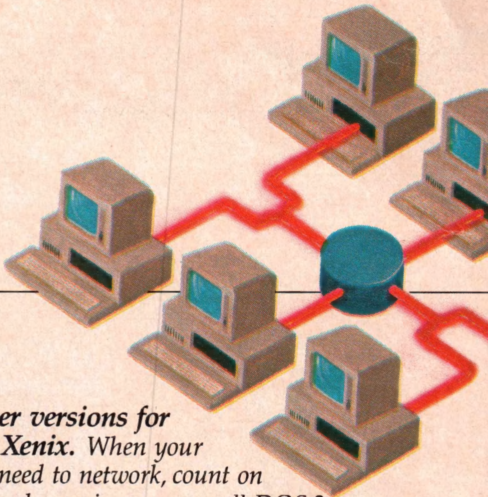
Information Services, P.O. Box 20212
5000 Arlington Centre Blvd., Columbus, OH 43220

An H&R Block Company

Circle **no. 237** on reader service card.

Btrieve®

The Programmer's Choice.



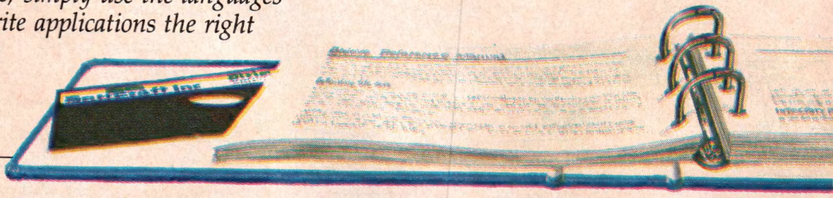
When you're serious about application development, there's just one choice for file management: Btrieve. With what *Computer Language* calls "near mainframe functionality[1]", Btrieve sets the file management standard for PC applications. With Btrieve loaded in your PC, your programs can use simple subroutine calls to retrieve, store and update records.

Multi-user versions for LANs and Xenix. *When your applications need to network, count on Btrieve. A single version runs on all DOS 3 LANs, including IBM PC Network and Novell Advanced Netware. Btrieve is also available for Xenix and multitasking operating systems such as MultiLink Advanced, Microsoft Windows and IBM Topview.*

B-tree based for high performance. *Performance is all-important, especially as your database grows. That's why Btrieve implements the b-tree file structure—the fastest, most efficient method of accessing data.*

Interfaces to C, BASIC, Pascal, COBOL. *Don't waste time programming in awkward fourth generation languages! With Btrieve, simply use the languages you know best—and write applications the right way. Over 15 language interfaces available.*

Built-in security features. *Lock up sensitive data with Btrieve's password protection and unique data encryption scheme—especially useful in local area networks.*



Help is just a phone call away. *Need technical support? You've got it! Btrieve users receive 30 days of unlimited phone support at no charge. This "Direct Connect" policy is renewable for a full year at low cost. And try SoftCraft's free bulletin board for technical tips, seven days a week.*

Thorough documentation, easy implementation. *Getting started with Btrieve is easy: the manual is packed with examples of every Btrieve function in BASIC, Pascal, COBOL and C.*

Database queries, report writing. *Add Xtrieve™ to your Btrieve applications for a fully-relational DBMS. Xtrieve's menu-driven interface gives your users the on-line query capabilities they need—without programming. Add our report writer option to produce custom reports and forms.*

No royalties. *Need we say more?*

Fault tolerant. *Btrieve insures against database disasters. Two levels of fault tolerance guarantee data integrity during accidents or power failures—even if lightning strikes. No extra programming required.*

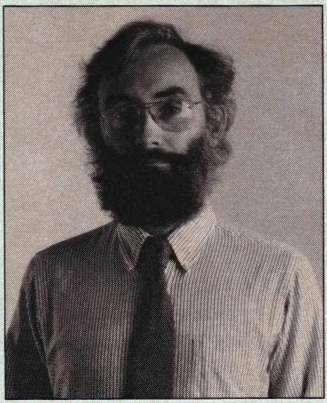
SoftCraft

P.O. Box 9802 #917 Austin, Texas 78766 (512) 346-8380 Telex 358 200

Suggested retail prices: Btrieve, $245; multi-user Btrieve, $595; Xtrieve, $245; multi-user Xtrieve, $595 (for report generation, add $145 for single-user and $345 for multi-user). Available from SoftCraft and selected distributors. Requires PC-DOS or MS-DOS 2.X, 3.X, Xenix. Btrieve is a registered trademark and Xtrieve is a trademark of SoftCraft Inc. [1]From Computer Language, *November 1985.*

Circle **no. 40** on reader service card.

EDITORIAL



Looking back, I can see that it was shortly after I started commuting from Santa Cruz into Silicon Valley that I began to see the importance of the bandwidth problem. Squeezing in with thousands of other commuters through the mountain pass to fight my way across Silicon Valley and up crowded Highway 101 got me thinking about channel capacity.

There are interesting similarities and relationships between transportation bottlenecks and the information-transmission bottlenecks that I believe represent the greatest challenge for programmers in the next decade. Just as data can be organized into packets, commuters can be packed up in trains or car pools to increase their transmission rate and decrease the frequency of collisions. The vehicular capacity of a highway really is a form of channel bandwidth. And anyone who has fallen into a web of one-way streets such as Berkeley's has something to say about network topology.

Relationships between transportation and communication are important when you consider trading one off against the other. At *DDJ* we have been, in a limited way, exploring the benefits of telecommuting. (I'm referring to the exchange of the costly movement of bodies for the lower-cost transmission of bits—not driving with a telephone at your ear, which so many of my fellow commuters are doing and which could also be called telecommuting.) I'm skeptical about how closely people can work together at physical long distance, but I'm willing to experiment because the cost of transportation, particularly the cost in time, is so great.

That cost will not go down, the world's transportation bottlenecks are only going to get worse, and there is little to indicate that anyone is thinking very hard about what technological fixes there may be.

It would seem that we are closer to solutions to communications bottlenecks: we have seen the development of communications satellites, computer networks, and sophisticated routing software for voice and data. The picture phones of science fiction exist today. (See Swaine's Flames, page 152.)

In contrast, science fiction's visions of future transportation are further from realization. I did recently drive a car equipped with an impressive computerized navigation system from Etak Inc., of Menlo Park, but innovation like this in the area of consumer transportation is rare. Cars are still cars, highways are still highways, and traffic is still a battleground of a thousand conflicting desires. Maybe society can no longer afford the luxury of our current driving habits in constricted channels like Highway 101, and cars and highways should be modified to permit centralized control and routing. It's not such a radically ambitious notion, given the context of Star Wars, and it's a lot more down-to-earth.

This little peninsula with its runaway population growth and its wealth of technical expertise seems like the ideal place for the experiment to begin.

Staccato signals of constant information. . . .
These are the days of miracle and wonder
This is the long distance call
—Paul Simon

Michael Swaine

Michael Swaine
editor-in-chief

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Michael Swaine*
Editor *Nick Turner*
Managing Editor *Vince Leone*
Assistant Editors *Sara Noah Ruddy*, *Levi Thomas*
Technical Editor *Allen Holub*
Contributing Editors *Ray Duncan*, *Michael Ham*, *Bela Lubkin*, *Namir Shammas*, *Ernest R. Tello*
Copy Editor *Rhoda Simmons*

Production
Production Manager *Bob Wynne*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Typesetter *Jean Aring*
Cover Photographer *Michael Carr*

Circulation
Circulation Director *Maureen Kaminski*
Newsstand Sales Mgr. *Stephanie Barber*
Book Marketing Mgr. *Jane Sharninghouse*
Circulation Coordinator *Kathleen Shay*

Administration
Finance Director *Kate Wheat*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura Di Lazzaro*

Advertising Director
Robert Horton *(415) 366-3600*

Account Managers
Lisa Boudreau *(415) 366-3600*
Gary George *(404) 897-1923*
Michele Perkins *(317) 875-8093*
Michael Wiener *(415) 366-3600*
Cynthia Zuck *(718) 499-9333*
Promotions/Srvcs.Mgr. *Anna Kittleson*
Advertising Coordinator *Charles Shively*

M&T Publishing Inc.
Chairman of the Board *Otmar Weber*
Director *C.F. von Quadt*
President and Publisher *Laird Foshay*
Associate Publisher *Michael Swaine*

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submissions: Send manuscripts and disk (with article and listings) to the Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Requested: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX: 620430 (WUI).

People's Computer Company
Dr. Dobb's Journal of Software Tools is published by M&T Publishing Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Our thanks to NASA for supplying this computer enhanced ultraviolet photo taken by Skylab IV of a solar prominence reaching out 350,000 miles above the sun's surface

Genius Begins With A Great Idea ...

But The Idea Is Just The Beginning

What follows is the time consuming task of giving form and function to the idea.

That's why we concentrate on building into our software development systems functions and features that help you develop your software ideas in less time and with less effort.

We've started 1987 by releasing new versions of our MS-DOS, Macintosh, Amiga, ROM, and Apple // C development systems. Each system is packed with new features, impressive performance, and a little bit more genius.

Aztec C86 4.1
New PC/MS-DOS • CP/M-86 • ROM

Superior performance, a powerful new array of features and utilities, and pricing that is unmatched make the new Aztec C86 the first choice of serious software developers.

Aztec C86-p Professional System $199
• optimized C with near, far, huge, small, and large memory + Inline assembler + Inline 8087/80287 + ANSI support + Fast Float (32 bit) + optimization options • Manx Aztec 8086/80x86 macro assembler • Aztec overlay linker (large/small model) • source level debugger • object librarian • 3.x file sharing & locking • comprehensive libraries of UNIX, DOS, Screen, Graphics, and special run time routines.

Aztec C86-d Developer System $299
• includes all of Aztec C86-p • Unix utilities make, diff, grep • vi editor • 6 + memory models • Profiler.

Aztec C86-c Commercial System. $499
• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 support • One year of updates.

Aztec C86 Third Party Software

A large array of support software is available for Aztec C86. Call or write for information. The following is a list of the most requested products: Essential Graphics • C Essentials • C Utility Library • Greenleaf Com. • Greenleaf General • Halo • Panel • PC-lint • PforCe • Pre-C • Windows for C • Windows for Data C terp • db Vista • Phact • Plink86Plus • C-tree.

CP/M • TRS-80 • 8080/Z80 ROM

C compiler, 8080/Z80 assembler, linker, librarian, UNIX libraries, and specialized utilities.

Aztec C II-c (CP/M-80 & ROM). $349
Aztec CII-d (CP/M-80) $199
Aztec C80 (TRS-80 3&4) $199

Aztec C68k/Am 3.4
New Amiga Release

Amiga user groups across the USA voted Aztec C68k/Am release 3.3 the best Software Development System for the Amiga. Release 3.4 is more impressive.

Aztec C68k/Am-p Professional $199
A price/feature/performance miracle. System includes: optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Amiga libraries • examples.

Aztec C68k/Am-d Developer $299
The best of Manx, Amiga, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep and vi.

Aztec C68k/Am-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C68k/Mac 3.4
New Macintosh Release

For code quality, reliability, and solid professional features, Aztec C for the Macintosh is unbeatable. This new release includes features and functions not found in any other Macintosh C development system.

Aztec C68k/Mac-p Professional $199
• optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Macintosh libraries • examples.

Aztec C68k/Mac-d Developer. $299
The best of Manx, Macintosh, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep • vi editor.

Aztec C68k/Mac-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C65
New ProDOS Release

Aztec C65 is the only commercial quality C compiler for the Apple II. Aztec C65 includes C compiler, 6502/65C02 assembler, linker, library utility, UNIX libraries, special purpose libraries, shell development environment, and more. An impressive system.

Aztec C65-c Commercial $299
• runs under ProDOS • code for ProDOS or DOS 3.3

Aztec C65-d Developer. $199
• runs under DOS 3.3 • code for DOS 3.3

Aztec ROM Systems
6502/65C02 • 8080/Z80 • 8086/80x86 • 680x0

An IBM or Macintosh is not only a less expensive way to develop ROM code, it's better. Targets include the 6502/65C02, 8080/Z80, 8086/80x86, and 680x0.

Aztec C has an excellent reputation for producing compact high performance code. Our systems for under $1,000 outperform systems priced at over $10,000.

Initial Host Plus Target $750
Additional Targets $500
ROM Support Package $500

Vax, Sun, PDP-11 ROM HOSTS

Call for information on Vax, PDP-11, Sun and other host environments.

C' Prime
PC/MS-DOS • Macintosh
Apple II • TRS-80 • CP/M

These C development systems are unbeatable for the price. They are earlier versions of Aztec C that originally sold for as much as $500. Each system includes C compiler, assembler, linker, librarian, UNIX routines, and more. Special discounts are available for use as course material.

C' Prime $75

Aztec Cross Development Systems

Most Aztec C systems are available as cross development systems. Hosts include: PC/MS-DOS, Macintosh, CP/M, Vax, PDP-11, Sun, and others. Call for information and pricing.

How To Become An Aztec C User

To become a user call 800-221-0440. From NJ or international locations call 201-542-2121. Telex: 4995812 or FAX: 201-542-8386. C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Aztec C is available directly from Manx and from technically oriented computer and software stores. Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee.

Most systems are upgradable by paying the difference in price plus $10. Site licenses, OEM, educational, and multiple copy discounts are available.

To order or for more information call today.

1-800-221-0440

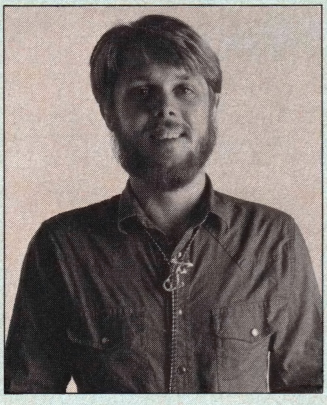
In NJ or international call (201) 542-2121 • TELEX: 4995812

MANX

Manx Software Systems
1 Industrial Way, Eatontown, NJ 07724

Circle **108** on reader service card.

RUNNING LIGHT



Several companies are feverishly (and secretly) working on new chips called "neural networks." These chips are designed to emulate various functions of the human brain. They contain thousands of standard microcircuit elements organized in a distinctly nonstandard way—hundreds of simulated neurons, each one of which is an always-active component capable of making simple decisions. Each "neuron" is connected by "synapses" to up to several hundred other neurons and can change its own activation potentials in response to signals it receives.

This could be the most important new invention in cybernetics since the Turing machine. Why? Because neural networks function in much the same way as your own brain (and mine, I think). Once they surpass a certain threshold of complexity, neural networks (whether they are actually implemented in hardware or simulated on traditional machines) begin to exhibit some interesting properties. Depending on their precise organization, they can act as spectacularly efficient pattern recognizers, solvers of multiple simultaneous equations (and other complex mathematical problems), learning machines that actually reach their own conclusions about the knowledge presented, and sophisticated device controllers—in short, all the functions that are currently performed by neural networks in your own body. A properly designed hardware neural network, for example, could easily provide a good solution to a 30-city traveling salesman problem in just a few (very few) microseconds. It's not guaranteed to be the best solution, but it will be extremely good. And the solution would be reached before a traditional computer had even initialized its variables. Watch these pages for more about this exciting new field.

This issue marks the introduction of a new theme for *DDJ*. As the speed and storage of computers increase more and more rapidly, it becomes increasingly important to be able to move large quantities of information from one computer to another. We have advanced fiber-optic cables and satellite microwave links, but we also have an exponential growth in the amount of information being sent. The problem is one of bandwidth—the capacity of the communication channels is not growing as rapidly as the need to communicate.

We'll be looking at the bandwidth problem from several different perspectives. In this issue Ronald White describes graphical quadtrees, a way to compress image data, sometimes dramatically. We also have a comparative review of four public-domain archiving programs. Archived files are files that have been compressed through the elimination of redundant data. Such files can be transmitted faster over modem lines, and because of this, archive programs have become popular on computer bulletin-board systems.

Do you have a project that excites you? Would you like to write about it for *DDJ*? Your first step is to give us a call at (415) 366-3600 and ask for a copy of the writers' guidelines. This wondrous document contains carefully refined advice that is intended to help you get published.

Nick Turner

Nick Turner
editor

ARCHIVES

Call for Innovation

"Think Big, dream bigger. Guest essays, dream pieces, signposts and other nefarious schemes for advancing this micro technology are welcome here, in addition to your hard software contributions to the community."—*Marlin Ouverson,* DDJ, *January 1982.*

WCCF

"The 6th West Coast Computer Faire is over (sigh of relief and aching feet). Director Jim Warren's roller skates have been hung up for another year, or at least until his rumoured mini-faires get under way. This year's was even bigger than expected . . . .

"Perhaps the biggest surprise to commercial concerns was the profile of the 'average' attendee. All the sales people had been coached well in advance on how to spill forth a technical-sounding pitch; none seemed able to answer basic questions like, 'Why should I have a computer?' 'What can I use it for?' 'I don't know anything about computers—where do I start?' Would-be consumers wandered from booth to booth in search of someone who could still remember how to speak their lingo and give a down-to-earth reply."—*Marlin Ouverson,* DDJ, *June 1981.*

Ten Years Ago in *DDJ*

"FAIRE CONSUMES EDITOR—DDJ LATE

"What more can we say? *DDJ*'s entrepid editor involved himself as chairperson of the First West Coast Computer Faire and discovered it to be an infinite sink of time. And he was spread much too thin before he started. Xeroxing editors failed."

"From time to time, over the past twenty years or so, there have been predictions that we will soon have inexpensive mass storage devices capable of holding the entire Library of Congress in a $19 desk-top unit. These chimerical devices are usually based on some far-out technology involving proton resonance, magnetic bubbles, or holograms. Well, I'm still waiting patiently for such a device to materialize but I'm not holding my breath."—*Jim Day,* DDJ, *March 1977.*

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia
Running Light Without Overbyte

A Challenge to Microsoft® C...

We challenge Microsoft C (Ver 4.0) to a C compiler duel to the finish, measuring compile, link, and execution times. If they win, we will stop advertising for two months.

by Roy Sherrill

If Microsoft C (Ver 4.0) can beat Optimum-C then we will stop advertising in all magazines for two full months and, win or lose, we will publish the results in its entirety. Even the Microsoft ads say "The Fastest C you've ever seen," so let the challenge begin.

Walter says Optimum-C is better

It all started when Walter Bright, the developer of Optimum-C, was explaining his new global optimizing C compiler and how it's code would be faster than Microsoft C (Ver. 4.0). Walter and I were frustrated because here we had a C compiler that would beat Microsoft C on 7 out of 10 benchmarks and also compile and link faster; yet our marketing consultant, Mark Astengo, told us that Microsoft C had a lock on the C compiler market and by 1990 they would probably have an 80% market share. Then Mark said, "Roy, if your C compiler is as fast as you say it is, why not challenge Microsoft C to a duel? If Microsoft wins, Datalight should stop advertising for two months and print the results of the test, win or lose." Well, I've always been one for a challenge. So here it is...

We only ask the following...

The benchmark suite will consist of the set of programs that Microsoft supplied to *Computer Language* for their February 1987 C compiler review issue. Microsoft will make available the programs to Datalight at least two weeks prior to the benchmarking. The benchmarking will be between Microsoft C 4.0 and Optimum-C. It will occur at a mutually agreed upon time and place. Interested individuals will be allowed to attend. The benchmarks will be compiled and run on a standard IBM PC-AT.

There will be two separate tests for each program: compile and link speed, and execution speed. For each test, a representative from each company will set up the compiler so that it performs at its best.

The benchmarks will be adjusted so that they take sufficiently long to run, that the tolerance involved in timing them is insignificant. The winner is determined by the compiler with the faster execution times for the majority ofthe benchmarks. We'd like an answer from Microsoft no later than April 1, 1987.

So what's a global optimizer?

A global optimizer looks at an entire function at once, analyzing and optimizing the whole function. A technique called data flow analysis is used by Optimum-C to gather information about each function. This enables your compute-bound programs to execute as much as 30% faster after global optimization. But, there is one catch...because the global optimizer ruthlessly searches for ways to speed-up execution speed and minimize memory usage, it has relatively slow compile times. No need to worry, though, because you can merely turn the global optimizer off. In fact, you can select all, none, or partial of the following optimizations: constant propagation, copy propagation, dead assignment elimination, dead variable elimination, dead code elimination, do register optimizations, global common subexpression elimination, loop invariant removal, loop induction variables, optimize for space, optimize for time, very busy expressions.

Choose from five memory models

Speed your programs by selecting the memory model that best suits your application.

Memory Models

Model	Code	Data
Compact	64k total code & data	
Small	64k	64k
Program	1M	64k
Data	64k	1M
Large	1M	1M

Compiling, one step...

Now with the one step DLC program you can create .OBJ, .EXE and .COM files. Also, DLC can handle multiple files and run MASM on your assembly files.

Try Optimum-C risk free

Try Optimum-C for 30 days and if you are not 100% satisfied return it for a full refund. Also, for a limited time we are including a *free C tutorial which is a combination workbook and floppy disk to help lead you through the C language with tutorials, quizzes, and program exercises.

O.K. Microsoft, it's up to you. We've put two months of advertising on the line that says you can't beat Optimum-C to a real test. Your answer, please?

PRICES

Developer's Kit	still only $99
Optimum-C (includes library source code)	$139

Add $5 for shipping in US/$15 outside US
COD (add $2.50)

Not Copy Protected

ORDER TOLL-FREE TODAY!

1-800-221-6630

Magazine Reviewers Shocked by DATALIGHT's Performance...

"Reviewing this compiler was quite a surprise for us. For such a low price, we were expecting a "lightweight" compiler. What we got was a package that is as good as or better than most of the "heavyweights." Datalight C implements a complete C language. It also compiles quickly, doesn't take up much disk space, and looks impressive in the benchmarks."

***DR. DOBBS*, August 1986**

"This is a sharp compiler!... what is impressive is that Datalight not only stole the compile time show completely, but had the fastest Fibonacci executable time and had excellent object file sizes to boot!"

***COMPUTER LANGUAGE*, February 1986**

Optimum-C Version 3.0

- Full UNIX System 5 C language plus ANSI extensions
- Fast/tight code via powerful optimizations including common sub-expression elimination
- DLC one-step compile/link program
- Multiple memory model support
- UNIX compatible library with PC functions
- Compatible with DOS linker and assembler
- Third-party library support
- Automatic generation of .COM files
- Supports DOS pathnames, wild cards, and Input/Output redirection
- Compatible with Lattice C version 2.x
- Interrupt handling in C
- Debugger support
- ROMable code support/start-up source

MS-DOS® Support Features

- Mouse support
- Sound support
- Fast screen I/O
- Interrupt handler

MAKE Maintenance Utility

- Macro definition support
- MS-DOS internal commands
- Inference rule support
- TOUCH date manager

Tools in Source Code

- cat—UNIX style "type"
- diff—Text file differences
- fgrep—fast text search
- pr—Page printer
- pwd—Print working directory
- wc—Word count

Datalight

Box 82441
Kenmore, Washington 98028
(206) 367-1803

Microsoft and MS-DOS are registered trademarks of the Microsoft Corporation.

*Limited offer available exclusively to readers who purchase directly from Datalight.

Circle no. 203 on reader service card.

LETTERS

Programming Ethics

Dear *DDJ*,

I've just read Allen Holub's December Viewpoint and was pleased to see him make his position known to the world. It's not an easy thing to do. I'm sure he'll receive letters criticizing his position.

On that note, allow me to introduce PeaceNet, an international computer network dedicated to improving communication between people worldwide on the issues of peace. PeaceNet is in need of programmer support, and anything readers can do, either directly or indirectly, would be helpful. PeaceNet is running on a Unix system (a Plexus P-60 [68000-based] to be exact) and is accessible through Telenet and via direct dial (Palo Alto, California).

In December, PeaceNet had more than 700 users and was gaining more than 100 per month. We have conferences on many important issues as well as "action alerts" and activity calendars. Readers interested in finding out more should call (415) 486-0624.

Corwin Nichols
223 Forest Ave.
Palo Alto, CA 94301

Dear *DDJ*,

I am responding to Allen Holub's Viewpoint in *DDJ*, December 1986. First, let me say, "bravo, Allen"; then let me say, "I disagree."

I say bravo because Allen states that he has examined the issues of working on defense contracts and that he feels he cannot live with his conscience if he performs this kind of work. He also states that "there are people who have thought about these issues and have come to the opposite conclusion." I am one of these people, hence I disagree. He feels these people are wrong but that they are acting according to their beliefs in working on defense projects. He says he has his problem with people who refuse to look at the issues and work on defense projects anyway. I agree.

I am one of the people who has examined the issues and feels that defense work is a necessary activity in this world today. Although I wish that we were living in that time when "they shall beat their swords into plowshares, and their spears into pruning hooks: nation shall not lift up sword against nation, neither shall they learn war anymore" (Isaiah 2:4), we have Armageddon to face between here and there. I agree that defense work is a grave-digging activity: the enemy's grave. Not to engage in defense is a grave-digging lack of activity: our own grave. I cannot think of abstinence from defense work as other than suicide.

Robert J. Brown, III
Elijah Laboratories International
5150 W. Copans Rd., #1135
Margate, FL 33063

Dear *DDJ*,

After reading Mr. Holub's article in the December issue of *DDJ*, I felt I could not remain silent. I must disagree with Mr. Holub's opinions about nuclear weapons and whether or not an engineer could change this.

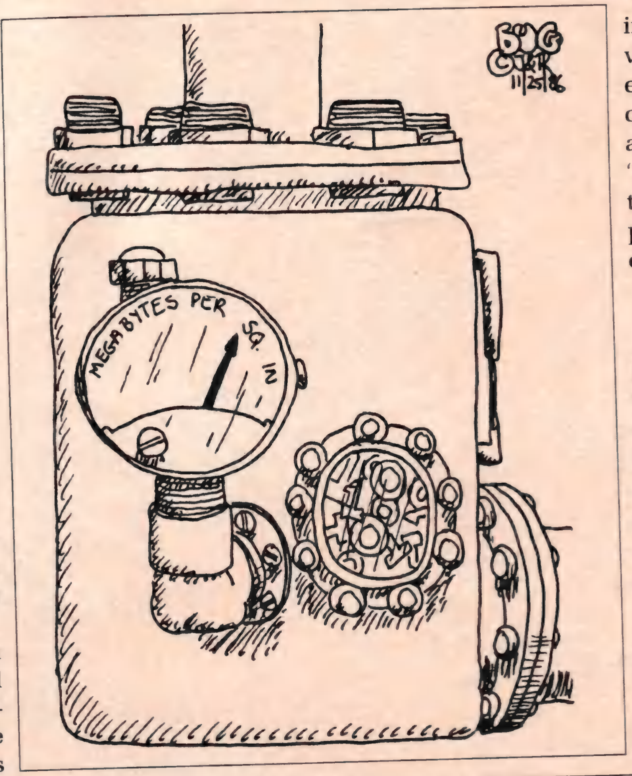
Assume just for the moment that all engineers agreed with him. Then what? Do we want a nonengineer cobbling together our weapons? Would that make us feel safer?

He said that if we didn't design them, they wouldn't exist. If all the ethical engineers in this country were to walk out on their defense contracts, do you know who would fill in for them? The unethical engineers.

I agree that we shouldn't sit around and wait for the bombs to be dropped. All of us, independent of our profession, should be much more politically aware—aware of the issues, who we vote for, and what we can do. That is the tack we need to follow as humans, not as engineers, to make the world a better, safer place to live.

I had a course in engineering ethics as well as an advanced philosophy course on ethics. I agree that these courses should be required of all engineering majors. The "tools" that you gain from these courses are just as important as everything else an engineer must learn.

Dave Podolske
University of Wisconsin



Dear *DDJ*,

Thanks to Allen Holub for writing his article about his personal decision not to work on weapons. I am gladdened whenever someone of his technical prowess speaks up on this issue.

He was brave to write it, and you were brave to print it.

You might tell your readers that there is an organization, Computer Professionals for

(continued from page 10)

Social Responsibility, that deals with these issues. Its address is Box 717, Palo Alto, CA 94301. Also, the ACM SIGSOFT and SIGAS publish related information.

Kerry Tatlow
1706 Charles
Rockford, IL 61108

Dear *DDJ*,
Thank you and congratulations for running Allen Holub's Viewpoint on programming ethics. It says a lot for *DDJ*'s position on the forefront of software technology in particular and thought in general, regardless of where one stands on the guns and butter debate.

Michael Gardner
Wordtech Systems, Inc.
P.O. Box 1747
Orinda, CA 94563

Dear *DDJ*,
I enjoyed reading Allen Holub's December article concerning the choices that engineers and programmers make in determining the effect of the work that they do upon society. It appears that we are primarily concerned with the intellectual challenge and financial rewards of our technical careers and that we seldom think about the effect of our work on society as a whole.

However, I think that the following is evidence that people are also concerned with the moral aspects of their work: There is a tendency for technical people working in the defense industry to be paid more than people doing nondefense-related work. A large part of the reason for the pay discrepancy is that many people just don't want to work on weapons. In order to make defense-industry work attractive enough to fill the positions, employers are forced to pay more than the market rate to overcome people's natural distaste.

Eric D. Andresen
529 Stone Dr.
Novato, CA 94947

In Search of a Sine

A number of people who responded to Richard Campbell's article "In Search of a Sine," published in the December 1986 issue, pointed out additional references for transcendental algorithms. Following is a list taken from the letters.—eds.

Abramowitz, M. I. A. Stefun ed., *Handbook of Mathematical Functions and Formulas, Graphs and Mathematical Tables.* National Bureau of Standards Applied Math Series, 55, Washington, D.C.: U.S. Govt. Printing Office, 1964.
Acton, Foreman S. *Numerical Methods That Work.* New York: Harper & Row, 1970.
Cody, W., and Waite, W. *Software Manual for the Elementary Functions*, Englewood Cliffs, N.J.: Prentice-Hall, 1980.
Hart, J., et al. *Computer Approximations.* New York: Wiley, 1968.
Hastings, Cecil, Jr. *Approximations for Digital Computers.* Princeton, N.J.: Princeton Univ. Press, 1955.
Knuth, D. E. *Art of Computer Programming, Volume 2: Seminumerical Analysis.* Reading, Mass.: Addison-Wesley, 1969.

Dear *DDJ*,
Regarding the article "In Search of a Sine," I would like to point out that the Taylor's series expansion, although of inestimable value in doing analytical, theoretical work, is in general of little value in computing numerical approximations of a function and is hardly ever used. There is a good reason for this. When expanding a function into its Taylor's series expansion, you choose a reference point upon which to anchor the expansion and then use the series to approximate the function in the vicinity of this point. If, for example, you are interested in approximating sin(x) (as Mr. Campbell was), you might choose the anchor point $x=0$ and come up with the result:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} \ldots$$

or, in expanded form:

$$\sin(x) = x - 0.166667x^3 + 0.0083333x^5 + \ldots$$

It is evident that this approximation produces an exact result at the reference point, $x=0$.

If, however, you choose the reference point $x=\pi/4$ radians (45°), the resultant series becomes:

$$\sin(x) = -0.00924739 + 1.04438x - 0.0758732x^2 - 0.117851x^3 + \ldots$$

This approximation isn't worth beans for $x=0$ but is exact at the reference point—well, exact to within the accuracy of the coefficients, as represented, anyway.

The point is this. The Taylor's series approximation of a function—any function—is exact at the reference point about which it is expanded but gets worse the farther you get from this point. Furthermore, it often gets worse fast! To achieve good overall accuracy, you would have to use many series, each anchored at an appropriate point.

A better way was invented by a clever mathematician named Chebyschev (sometimes spelled Tchebyschev or variant thereof). Instead of approximating a function with a series of powers of x, he whipped up some nifty polynomials in x (called Chebyschev polynomials, what else?) and approximated the function with a series of these polynomials. Of course, all the powers of x in these polynomials eventually combine to produce something looking very much like a Taylor's series, except that the coefficients are slightly different and they have the absolutely lovely property of distributing and bounding the approximation error over the range of the approximation. Not only that, the Chebyschev scheme produces approximations of comparable error to Taylor's but with fewer terms. The process is ofttimes called Chebyschev economization. Without going into the details of why this happens or how to do it (which would require a lengthy article in itself), suffice it to say that if you expand $\sin((\pi/2)x)$ in terms of Chebyshev polynomials and then simplify it, the result is:

$$\mathrm{Sin}((\pi/2)x) = 1.5706268x - 0.6432292x^3 + 0.0727102x^5$$

I have used the notation Sin( ) to mean an approximation of sin( ). This function has a maximum error of about 0.0001 over the range $-1 < x$

You'd Be Surprised Who Uses Essential Graphics Functions

And some of our well-known customers would be just as surprised if they saw themselves in this ad. We don't splash their names all over the place as a matter of professional courtesy.

Let's face it. Just because someone else uses a product is not reason enough to buy it. The clincher is that our programs run a *documented 40% faster* than the closest competitor. To complete the picture, our code is up to *75% smaller* due to efficient coding and the granularity of functions.

However, should curiosity get the best of you, call us at 201-762-6965 and we'll drop a few highly impressive names on you.

Behind every great program is a great library.

Draw Your Own Conclusions

When you're responsible for a project that includes advanced graphics, "graphics windowing," or character font manipulation, Essential Graphics is the clear choice. We've taken the grind out of graphics programming and replaced it with speed and versatility.

No Royalties, 30 Day Guarantee

We believe that selling you a programming tool does not make us your co-authors. So we don't charge any royalties or run time fees. If within 30 days you don't find our library satisfactory, dump the whole thing and receive a complete refund.

Functions At A Glance

Features:
- Fastest functions available
- Dots, Lines, Circles, Arcs, Pies, Bars
- Manipulate character fonts
- Move blocks, do animation
- User definable patterns
- Seed filling in a boundary
- Clipping on screen coordinates

Devices Supported:
- IBM, Epson, Oki printers
- HP Plotters, HP Laser Jet
- Microsoft, Logitech Mice

Graphics Adapters:
- IBM Color Graphics
- IBM Enhanced Graphics
- Hercules Graphics
- AT&T, Olivetti Graphics
- Tecmar Graphics Master
- Others (Call)

Compiler Compatibility:
- Microsoft, C, Fortran, Pascal
- Lattice C, Aztec C
- Computer Innovations C86, DeSmet C
- Wizard C, Mark Williams

$250.00

Do Your Homework

The library you buy can influence the rest of your programming life. We encourage you to do some checking before making a decision. When you've done your homework, you'll choose Essential. Call our support staff of experienced C programmers and find out before you buy how things will be after your check clears.

To order or for support call: 201-762-6965

For foreign orders contact:

England: Gray Matter Tel. (0364) 53499
Japan: Lifeboat Inc. of Japan Tel: 293 4711
West Germany: Omnitex Tel. 07623-61820

Essential Software, Inc.

P.O. Box 1003, Maplewood, New Jersey 07040

(continued from page 12)

< 1, which covers the entire first and fourth quadrants. If you require greater accuracy, you can use the approximation:

$$\mathrm{Sin}((\pi/2)x) = 1.57079631847x - 0.64596371106x^3 + 0.07968967928x^5 - 0.00467376557x^7 + 0.00015148419x^9$$

This approximation is valid over the same range and has a maximum error, which occurs at several places within the range, of about 0.000000005.

Another mathematician named Padé generated a slick way to do the job using the ratio of two polynomials. I'll say nothing more about this technique, except that it works sublimely with certain types of functions.

I would add one last thing in passing. You should never evaluate the polynomial (for example):

$$a + bx + cx^2 + dx^3$$

as it stands. You should always arrange it into the nested form:

$$((dx + c)x + b)x + a$$

This form requires fewer multiply instructions, thereby executing faster and producing smaller numerical errors.

Charlie Rose
Ball Aerospace Systems
P.O. Box 1062
Boulder, CO 80306

Dear *DDJ*,
The sine routine given by Richard Campbell in your December issue could be improved. One improvement would be to compute aa=a*a and then compute the sine approximation as:

```
s = (((C4 * aa + C3) * aa + C2) * aa
                        + C1 * a
```

using the same coefficients as before. This nested form of the polynomial accumulates the small terms first and thus reduces the errors due to floating-point rounding. By initially squaring a, you end up doing three additions and five multiplications.

Another improvement would be to use a ninth-order polynomial instead of a seventh-order one. The sine would then be computed as:

```
s = ((((C5 * aa + C4) * aa + C3) * aa
                 +C2) * aa + C1) * aa
```

where the coefficients are:

```
C1 = 1.5707963
C2 = -0.64596371
C3 = 0.079689679
C4 = -0.0046737666
C5 = 0.00015148513
```

The coefficients published in the December *DDJ* can be found on page 203, item SIN 3340, in *Computer Approximations* (J. Hart et al.), but are given to more precision. The coefficients for the ninth-order polynomial listed above can be found on page 204, item SIN 3341, of the same reference.

A third way to improve the routine would be to use the methods outlined in the two references and develop a program whose accuracy is limited only by the precision of the floating-point representation of the final result.

Harry J. Smith
Litton Computer Services
1300 Villa St.
Mountain View, CA 94039

Dear *DDJ*,
Here is some feedback on "In Search of a Sine." Some readers may notice that the coefficients C1 through C4 are somewhat different from the theoretical coefficients for the Maclaurin's series for the sine function. After truncating a series to a specific number of terms, it is advantageous to cook the coefficients using least-squares curve fitting. Presumably the values given for C1 through C4 were derived in this manner. Rearranging the sine formula before doing the actual computation would result in fewer operations. The BASIC rendition would be:

```
A2 = A*A
SIN = (((C4 * A2 + C3) * A2 + C2) *A2
                        + C1) * A
```

and the 32K assembly-language rendition would be:

```
DoSin
   MOVF F3,F1
   MULF F1,F1
   MOVF -0.004362469,F5
   MULF F1,F5
   ADDF 0.07948765,F5
   MULF F1,F5
   ADDF -0.645921,F5
   MULF F1,F5
   ADDF 1.570795,F5
   MULF F5,F3
   RET 0
```

Because the coefficients can be used only by the *DoSin* routine, there is no reason to keep them in a table. Making them immediate operands is more compact, faster, and more readable. When coding for the 32K, you often find that in-line coding is more compact than the looping method. When using the 32K, you must remain alert for opportunities to be liberated from your old habits.

For higher precision, you could further divide the range of A. That is, if A is more than 0.5, you take the cosine of (1.0 − A) using the Maclaurin's series for the cosine:

```
A2 = A*A
COS = (((D4 * A2 + D3) *A2 + D2) * A2
                    + D1) * A2 + 1.0
```

In order to realize the higher precision, you would need to recook the coefficients based upon the shorter range.

Another thing to keep in mind when doing math routines for the 32K is that the FPU is very fast; it does a double-precision multiply faster than the operands can be moved in and out of memory. Therefore, the old rule of thumb about the floating-point operation dominating the time is no longer true.

The bugs in the 32K alluded to by Mr. Campbell are almost certainly a thing of the past; if you have a reasonably mature version of the 32K, you should refrain from using any and all addressing modes with floating-point operands.

Neil R. Koozer
Kellogg Star Route Box 125
Oakland, OR 97462

DDJ

Some Very Impressive People Keep Our Asynch C Tools Under Their Hats

This is the only way we can get some of our customers to take their hats off to us in public. We understand. Most people like to keep a good thing to themselves.

Once you see how powerful and simple our functions are, you'll want to treat our Communications Library Plus as a well guarded trade secret too.

Essential provides the best alternative to Assembly language for communicating via an RS-232.

However, should curiosity get the best of you, call us at 201-762-6965 and we'll let you in on the identity of some of our secret admirers.

Behind every great program is a great library

What Good Are Library Functions If You Can't Get Them To Work?

Providing functions is not enough. Essential Communications Library Plus includes BreakOut, a slick on-line data monitor. BreakOut saves hours of frustration. We know, we used it to debug the XMODEM and other functions in Essential Communications.

No Royalties, 30-Day Guarantee

If within 30 days you don't find our library or BreakOut totally satisfactory, hang the whole thing up and receive a complete refund.

Functions At A Glance

- Interrupt driven to 9600 baud
- Hayes compatible support
- 150 page manual—tutorial
- XMODEM, XON/XOFF support
- Timer/Keyboard functions
- Input buffers to 500K
- Source included
- Demo terminal program
- Demo BBS system
- All major compilers supported
- Ctrl-Break status

Comm Library Plus/BreakOut $250 **Comm Library $185**

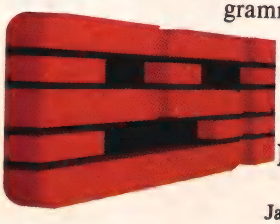
BreakOut On-line Monitor/Debugger

- Monitor RS232 ports up to 9600 baud
- Control comm variables and line signals
- Send/receive data using the scratch-pad editor
- Edit scratch-pad in Hex or ASCII
- Save data to file or re-transmit it
- User configurable keyboard macros
- Use symbols for control chars (<ACK>) for Hex 06

BreakOut $125

Do Your Homework

The library you buy will influence the rest of your programming life. When you've done your homework, you'll choose Essential. Call our support staff of C programmers and find out now how things will be after your check clears.



To order or for support call: 201-762-6965

For foreign orders contact:
England: Gray Matter Tel. (0364) 53499
Japan: Lifeboat Inc. of Japan Tel: 293 4711
West Germany: Omnitex Tel. 07623-61820

Essential Software, Inc.
P.O. Box 1003, Maplewood, New Jersey 07040

Compressing Image Data With Quadtrees

by Ronald G. White

When a graphical image is represented by an efficient quadtree, the amount of data can be less than with other representations.

A quadtree is a tree data structure in which each node can have four child nodes under it (compared to a binary tree in which each node can have two children). Quadtrees can be used as an efficient representation of graphical information and offer some interesting features. For a graphical quadtree, each node represents a square area of the graphical image and each of its four children represents one quadrant, or one fourth, of its area. These subareas are defined by dividing the original area in two equal halves, left and right, and dividing each of these halves in half, top and bottom. Thus the subareas, or quadrants, are of equal size and are also squares. The root node, called the top node, represents the entire image and has under it four children each representing one fourth of the entire image. This process of division is repeated until each child represents only a single pixel of the original image. If the original image is not a square with the number of pixels on each side being a power of 2, the image has to be filled out to that size with a background color. The bottom nodes, each representing one pixel, are leaf nodes of the tree. (Data structure trees are upside down from actual trees because the root node is said to be on the top and the leaf nodes on the bottom.) All the nodes in the tree at the same distance from the root—that is, having the same number of nodes between them and the root—are said to be on the same level. For graphical quadtrees, the levels are numbered starting from the bottom nodes—the leaves—which are at level 0. If the image is N×N pixels, where $N=2^k$, then the root, or top, node is at level k and the number of levels is k+1.

To represent a graphical image, the leaves of the quadtree need to have information about the corresponding pixel associated with them. If the image is a color image, then the bottom nodes will have a color value. This value can be an index into a color map, a set of RGB values, or some other color information. The information associated with nodes that are not at the bottom level has a less obvious meaning. If all four children of a nonleaf node are leaf nodes and they all have the same color, the parent node can adopt that color. In fact, the child nodes then become redundant and can be removed from the quadtree. With all four children removed, this node now becomes a leaf node even though it is not a level 0 node. Removal of unnecessary nodes is called pruning and can be repeated for successively higher levels, moving from the leaves toward the root.

If fewer than four child nodes have the same color, the parent can still adopt the predominant color of the children. If the children with the same color are leaf nodes, they can be removed from the tree. The parent node does not become a leaf node in this case because there are still child nodes under it, but you have reduced the size of the tree by removing at least some of the child nodes. It would be possible, even if all four child nodes were different colors, for the parent node to adopt one of the child nodes' colors and to remove that child node. As you will see later, in the section about locational codes, in the end this removal does not gain anything. You can get rid of one child node, but you then have to keep the parent

Ronald G. White, 161 S. 35th St., Boulder, CO 80303. Ronald has an M.S. in computer science from the University of Colorado. He is currently involved in porting software to new graphics terminals and workstations.

node. In my code I prefer to have the parent node adopt a color only if at least two of the child nodes have that color.

When a graphical image is represented by an efficient form of a quadtree (locational codes, presented in the next section, are one way), the amount of data can be less than with some other common representations such as simple pixel dumps or run-length encodings. This depends on the image, of course—a very fine mesh checkerboard pattern would not be represented efficiently by a quadtree. Images in which large areas are the same color can be represented by small quadtrees because higher-level nodes in the quadtree can represent large sections of the image with no need for lower-level nodes below them.

Another advantage of quadtrees is that if they are transmitted or displayed starting from the top level, then each successive level represents a closer approximation to the final image. This is particularly useful on some newer graphics displays that support a fast polygon fill command so that the area represented by a node can be filled quickly by a single command. Although I don't define the color of intermediate nodes in this way, each nonleaf node's color could be the average color of the area it represents. Each lower level would then provide a more accurate description of the image as it was transmitted or displayed. The method I use defines colors for intermediate nodes only if at least half of the final area represented by that node will be that color. In this case, each lower level provides additional information about some areas of the image that are not yet accurately defined. In an interactive situation where transmission of data is slow or costly, the user could be allowed to stop the transmission or display as soon as the image was accurate enough to be recognized as wrong or of no interest. With normal scan-line display of images, much of the image, and thus a lot of data, must be displayed before the contents of the picture can be guessed.

Quadtrees also have advantages for certain types of analysis and manipulation of images, but a discussion of these is beyond the scope of this article. For those of you who want to pursue this topic further, I've provided a list of references at the end of this article.

Locational Codes

The quadtree representation is efficient for certain types of processing, but it is not very efficient in terms of storage. Locational codes are a way of indicating the position of a node in the quadtree without actually storing the pointers from the root node to the given node. Instead, the path from the root node to the given node is coded as a single number. This is generally done by equating each direction—NW, NE, SW, or SE—from the parent to the relevant child as a single digit (for example, 0, 1, 2, and 3) and combining the digits representing the path into a single number. For example, the path NE-SE-SE-NW could be expressed as 1330, where the *1* represents the NE child of the root node, the first *3* represents the SE child of that node, the second *3* represents the SE child of the *13* node, and the 0 represents the NW child of the *133* node. Each node, then, has a unique representation. Using a base 10 representation, however, wastes storage space. In my code, I pack each direction value into two bits—that is, I use a base 4 representation. Other authors recommend using a base 5 representation so that the directions are the values 1, 2, 3, and 4 and 0 is reserved as a beginning marker. This is useful because, depending on the level of the node, the number of direction values—that is, digits—will vary. In base 5, the preceding example would be 02441, where the 0 indicates the beginning of the code. In base 4, there is no unique bit pattern to use as a beginning marker because all four possible two-bit patterns are used as direction values.

The disadvantage of the base 5 scheme is that it requires multiplying and dividing by five in order to manipulate

COMPRESSING IMAGE DATA
(continued from page 17)

the encoding. I originally used a base 5 encoding but found the conversion from a quadtree path to a locational code (and back) too slow. To speed up this process, I switched to a base 4 encoding and replaced the multiplies and divides by shifts and bit operations, which are faster. This representation also needs fewer bits to store it. To solve the problem of a beginning marker, I chose to mark the beginning with a 01 bit pattern. Because I always put a 01 in front of the actual direction values, I can search the bits in the locational code from left to right, two at a time, and know that the first nonzero pair is not a valid direction value but that the next pair is. Without this marker, it would be impossible to determine where the direction values start.

Because pointers are not needed with locational-coded quadtrees, nodes that only supply redundant information—that is, serve only as placeholders in the tree—can be removed, or pruned, from this form of the quadtree for more efficient storage or transmission. This pruning can result in significant space savings. Consider, for example, an image consisting of a single red pixel against a black background. In the pointer form of the quadtree, the root node would have a color of black because this is the predominant color of the area it represents (though not the only color). Three of its child nodes, representing the three quadrants not containing the red pixel, are not necessary because the information they would hold—the color black—is already held by their parent and all nodes under them would also have the same information. The fourth node, on the other hand, is necessary because somewhere at the bottom of its subtree is a node representing a single pixel and its information is different—that is, it has a color of red. This pattern—three nodes are unnecessary but the fourth is needed—is repeated down through the levels of the quadtree until you reach the node representing the red pixel. The nodes in the tree between the root node and the bottom node do not contain useful color information, but they are necessary to save the path from the root node to the bottom node. With the locational code form of the quadtree, each node is represented by a pair of numbers—the locational code itself and a color. For the image of a single red pixel, you need to have only two nodes—the root node and the single bottom node—all the intermediate nodes can be thrown away.

Another advantage of some locational codes is that when the codes are sorted into numerical order, the higher-level nodes come before lower-level nodes. This is true for the three coding schemes already mentioned (base 10, base 5, and my base 4 with special marker) because each lower-level code requires an additional digit to represent the next direction value. Sorted in normal numeric order, the higher-level nodes, having fewer digits, will always precede the lower-level nodes. The base 4 scheme I use preserves this feature because the 1 in front of the actual direction values is shifted two bits left at each lower level. My code, however, makes sorting unnecessary because it outputs the nodes from the root node one level at a time.

Listing One

Listing One, page 40, is a set of routines for converting a graphical image from pixels to a quadtree and outputting the quadtree in locational code form. I have not provided a main routine because initialization is likely to be application specific and possibly system dependent. The main routine should do whatever is necessary to make available a graphical image. Necessary tasks might involve reading the image into an array from a file, getting information interactively about what image or what part of the image the user wants, or initializing the display with the image if the image is to be taken directly from the display. The main routine also needs to perform initialization for the output of the quadtree. This could be opening a disk file and writing some header information to it, or it might involve opening a communications channel of some sort. After all this is done, the main routine calls *px2quad()*.

The only externally accessible routine is *px2quad()*, which must be passed the size of the image. If the actual size of the image is not a square with each side equal to a power of 2, the size passed to *px2quad()* is the smallest such square that the image will fit inside. These routines assume the availability of two routines, *getpix()* and *putlcc()*, that they can call. Besides the main routine, you must also supply these routines. *Getpix()* returns the color value of a pixel at a given x,y position, and *putlcc()* is called to output the locational code and color for each node.

The primary data structure used by the routines in Listing One, defined at the beginning of the listing, is used for each node in the quadtree. The first field, *child*, is an array for the four pointers to child nodes. The direction value serves as a subscript into this array, with the following correspondences: 0, 1, 2, and 3 correspond to the directions NW, NE, SW, and SE, respectively. The second field in the node data structure, *next*, is a pointer to the next node on the linked list used during output. This linked list is explained later as part of the explanation of

"Microport is the *fastest* path between *real* UNIX® System V and you."

Dimitri Rotow runs a company on the go. He has little time for hardware or software that doesn't do the ultimate job for his company's customers.

Two months ago he assessed three readily-available UNIX (and UNIX-like) operating systems. After evaluating pricing, packaging, documentation, quality, service, manufacturer's support, compatibility, third-party support, features, conformity and performance, he and his VP of engineering came to one conclusion: "To praise Microport is to praise the AT&T/Intel joint venture, because Microport's version of UNIX System V is virtually *identical*. However, more *value* is received since more middlemen are removed—that's much closer to buying direct and selling direct."

Bell Technologies' customers appreciate real value plus real System V. If you're an OEM, a reseller, an end-user or simply curious about our product—call us today and see what all the endorsements are about.

Microport Systems, Inc.
10 Victor Square
Scotts Valley, CA 95066

408/438-UNIX (438-8649, local)
800/822-UNIX (Inside California)
800/722-UNIX (Outside California)
FAX: 408/438-2511
Telex: 249554

MICROPORT

Team Use for The AT.

(continued from page 18)

the routines *outtree()* and *outnode()*. The next field, *color*, is the color associated with the node. I store this as a single value because I am using the index value into a color table. More extensive information could be substituted for the single value although this would complicate the code somewhat. The *ntype* field is a flag that indicates what type of node this one is. The three node types are defined in the next section of code. The *ntype* field is not absolutely necessary (as is explained later), but I find it useful. The final field in the node structure, *locode*, is the locational code for this node, which is calculated as each node is added to the quadtree.

The next section of code defines the three node types—*LEAF*, *BLEND*, and *WASH*. *LEAF* type nodes are nodes that have no further nodes under them. The color value of a leaf node represents the color of all the pixels in the area defined by that node even if it is not a bottom-level node. If, during the initial pruning of the quadtree, a node has two or three children that are leaf nodes and that share the same color, the parent node adopts the color of these child nodes and these nodes are removed from the tree. This parent node is marked as a *BLEND* type node to indicate that its color value is the color of pixels in areas represented by missing child nodes and that the node has child nodes that were not removed. All nodes that are neither *LEAF* type nodes nor *BLEND* type nodes are marked as *WASH* nodes, indicating that their only purpose is as placeholders for the pointers to child nodes and that their own color is not relevant.

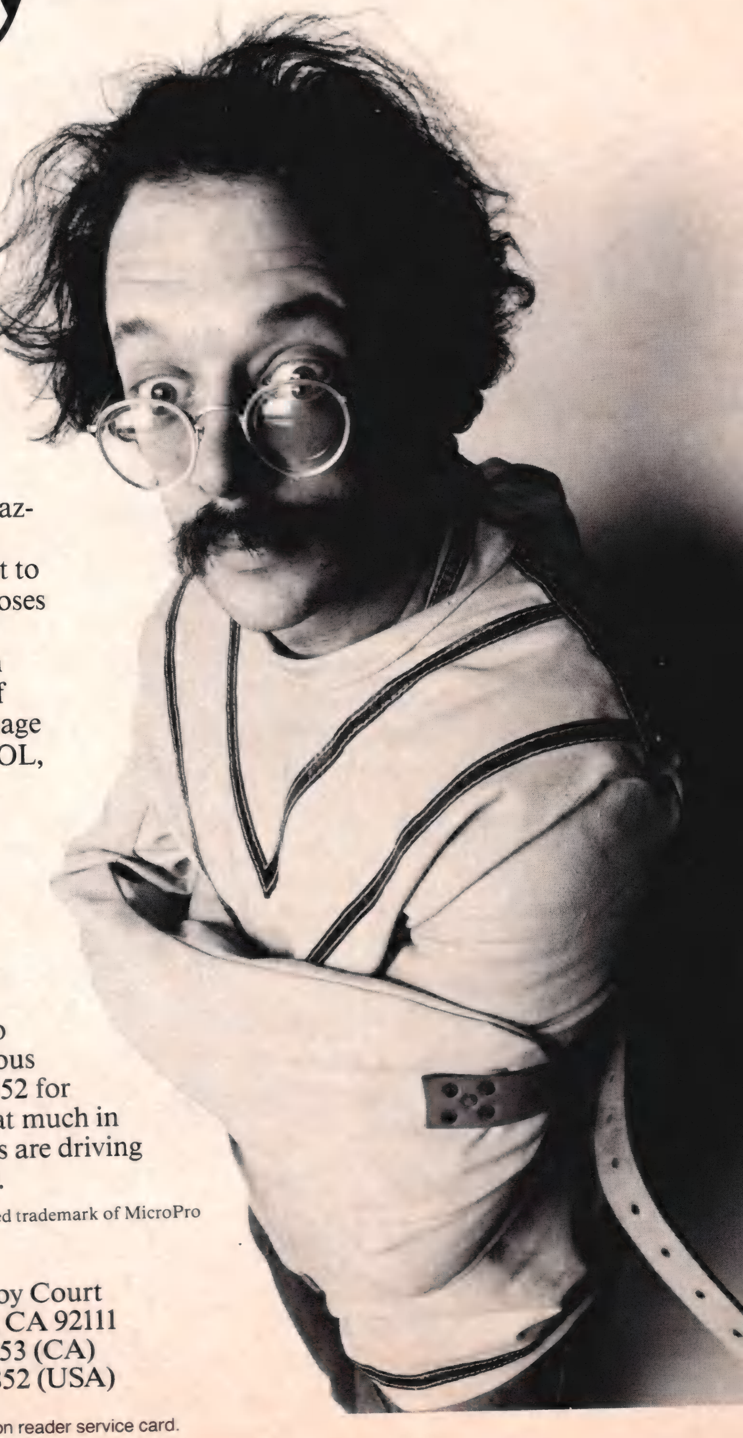
The routine *px2quad()* is the control routine for all the processing necessary to build and output the quadtree. The first thing it does is create the root node with a call to *crtnode()*. It then calculates the level number of the root node from the parameter *size*, which the calling routine passed to it. What this code does is find k, where $size=2^k$. (There may be a more direct way to do this.) After setting the locational code for the root to 0 (remember that the 1 is not actually part of the locational code but a beginning marker), *px2quad()* calls *addnode()* to add the root node to the currently empty quadtree. *Addnode()* calls *crtnode()* and itself recursively to create the rest of the quadtree. When *addnode()* finally returns, the quadtree has been built and the initial pruning done. *Px2quad()* then calls *outtree()* to control the final pruning and the output of the quadtree as locational codes and colors.

The function *crtnode()* creates a new node, initializes it with default values, and returns a pointer to it. I use a call to the system routine *malloc()* to get enough space for one node. This step could be a problem for several reasons. The first is the overhead associated with making this call perhaps hundreds of thousands of times during the creation of the quadtree. This overhead could be reduced by getting larger chunks of memory (or even statically allocating a very large array) and having *crtnode()* and *relnode()* maintain a list of free nodes. This approach complicates the code, of course, but it might be worthwhile for improved performance. The second potential problem is the amount of memory required for a quadtree that represents a reasonable-size image. I'll discuss this in more detail at the end of the article. *Malloc()* returns a *NULL* pointer if no more memory is available or something else goes wrong; *crtnode()* checks for this condition. The last part of the code initializes the newly created node. The node type is set to *LEAF*, indicating that this node does not have child nodes after it. If *addnode()*, the next routine, creates new nodes under this one, it will also call *condense()*, which, among other things, resets the node type.

Addnode() adds a newly created node to the quadtree. If this node is not at the bottom level, it creates four new child nodes under the current one with calls to *crtnode()* and calculates locational codes for these new nodes. This calculation is simple. The locational code for the current node is available in the node data. The locational code for a child node needs only one direction value added onto the end of the parent's code. Because the direction value is the same as the subscript into the array of pointers to the four child nodes, all the code has to do is shift the parent's locational code left two bits and add in the direction value for the child. *Addnode()* then calls itself recursively for each of the new child nodes. After these four new nodes have been added, *addnode()* calls the routine *condense()*, which examines the four nodes under the current node to see if any of them can be removed. If, on the other hand, the current node is at the bottom level—that is, this node represents a single pixel of the image—*getcolor()* is called to get the color of the pixel.

Condense() is the routine that does the initial pruning of the quadtree and is probably the most complex routine presented here. It first loops through the four children of the current node (*addnode()* does not call *condense()* for leaf nodes, so the current node will always have four children), collecting information about their colors. What the code is looking for is a predominant color—that is, a color shared by two or more child nodes. Nodes that are marked as type *WASH* are ignored because their color is meaningless (as is explained later). *Condense()* sets the current node's color to the predominant color. If no two children have the same color, then the next section is skipped. Otherwise, the code again loops over the four children. If a child had a type of *WASH*, it can't be deleted, so it is ignored. If a child has the same color as the predominant color, it is either removed, if it is a leaf, or demoted to type *WASH* otherwise. The reason why it can be removed if it is a leaf is that the area it represents is included in the area represented by its parent and the parent now has the same color—the color of the child node is therefore redundant. If a child node is not a leaf, this means that it has at least one child node of a different color below it and so it cannot be deleted without losing the pointers to its children. Its color, however, is now redundant information, and this fact is noted by marking it as a *WASH* type. The final section of code in *condense()* resets the node type of the current node. If all four children have been released (because they were all the same color and now this node has that color), then this node becomes a leaf node and is marked *LEAF*. If this node has adopted a color because two or more of its children have been removed or marked as *WASH* type nodes, it is marked as a *BLEND* type node to indicate that it is not a leaf node but its color is relevant information. If the node is

How a magical Genie saved this programmer from going crazy



File format changes were driving me nuts. Then, I got stuck with a 12 year old word processing file that had to be converted to WordStar® ASAP. I must have passed out momentarily, because the next thing I remember is a guy in a turban, jamming a diskette into my PC. He pressed a few keys and "Presto," he created a new utility for my program. The file conversion underway, I passed out again and when I woke up, he was gone—but the amazing utility software he had used remained.

File Genie, that's the name on the diskette. An amazing tool. Since then, I discovered that it will convert word processing and data files from most any format to any other. And, as if that isn't enough, it also: diagnoses and fixes errors in broken database files, instantly searches and replaces on seven and eight bit data (on ambiguous file names, with the most complete set of regular expressions I've ever seen), has a script language with IF, ELSE, WHILE, FULL SCREEN CONTROL, and the ability to run DOS commands or programs, comes with on-line context sensitive help, et cetera, et cetera. Allows printing of files without my printer reacting to control codes too.

With **File Genie** I write my own utility programs quickly and easily. As you can imagine, I guarded this little gem with my life, keeping it totally secret. Until everybody kept asking me how I was doing all these incredible things. I have to admit, I am sold. Which gets me to the really fabulous news. **File Genie** is available by calling 1-800-822-0852 for an introductory price of only $69.95. You'll save that much in aggravation the first time you use it. So if those files are driving you crazy, too, call and order your **File Genie** today.

WordStar® is a registered trademark of MicroPro

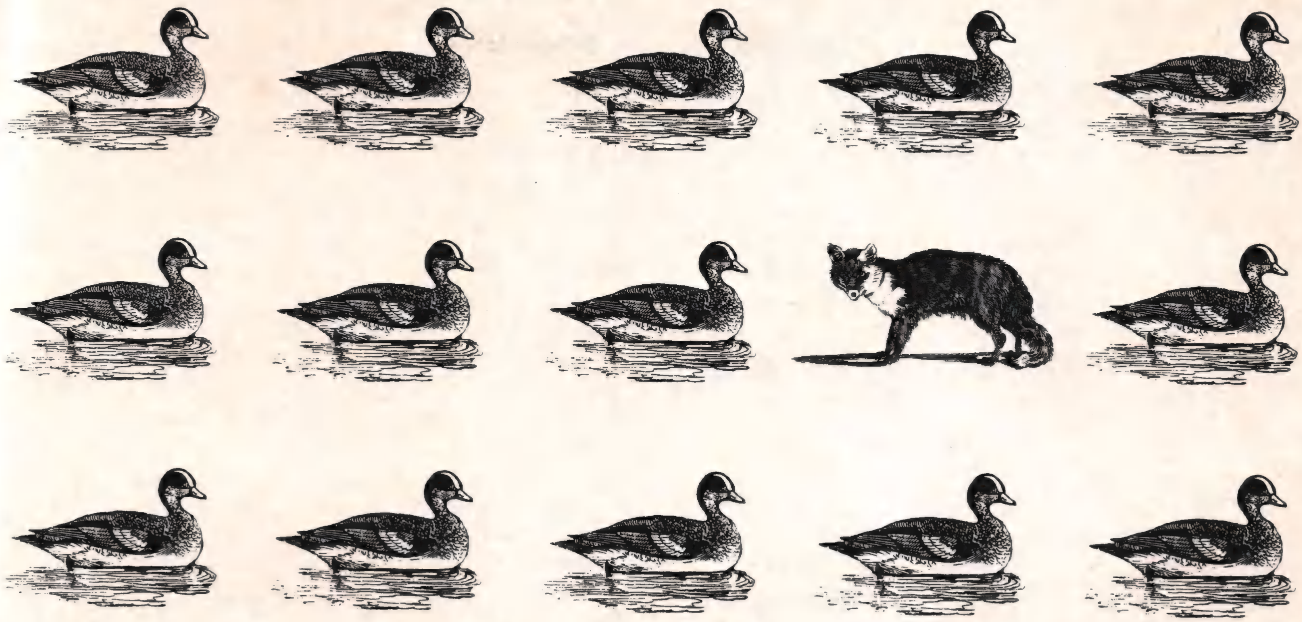
FILE GENIE™

A software product of Team Austin Inc.

6809 Convoy Court
San Diego, CA 92111
619-278-5353 (CA)
800-822-0852 (USA)

Circle **no. 204** on reader service card.

Turbo Programmers:



Find the culprit fast with T-DebugPLUS™

You're programming, getting all your ducks in a row, but there's a culprit in there, a bug. T-DebugPLUS, the new symbolic run-time debugger, helps you find that bug and fix it faster than ever before.

IN COMMAND. IN CONTROL.

See what happens *as* your Turbo Pascal programs run...examine variables...change values...be in control. The top half of your screen displays the source code, current line highlighted. You debug interactively on the lower half. All variable types are accessible, local and global. Set breakpoints at a procedure, function, or statement number.

INSTANT GRATIFICATION.

T-DebugPLUS integrates invisibly, automatically loading Turbo Pascal. Switch instantly between the debug and output screen. Jump instantly to the editor to fix that bug.

NEW...DEBUG IN OVERLAYS.

Debug in overlays with release 1.04. Examine areas of memory and CPU registers. A MAP generation utility even lets you use an external debugger on your program. T-DebugPLUS is only $60.

"T-DebugPLUS: Don't program in Turbo Pascal without it."

Neil Rubenking
PC Magazine

MORE POWER. MORE PRODUCTIVITY.

Turbo EXTENDER™ helps you break the 64K barrier for both code and data; use all 640K. A source code converison program, make facility, and run-time routines make writing and compiling large programs dramatically easier. Turbo EXTENDER is only $85.

TurboPower Utilities™ supplies nine powerful programs. Four Pascal utilities, including the acclaimed Pascal Structure Analyzer and Execution Timer, help you write programs that are bug free, faster, and easier to document. Five PC-DOS utilities help you analyze, change, and find your program files. TurboPower Utilities is only $55 for executable, $95 with source.

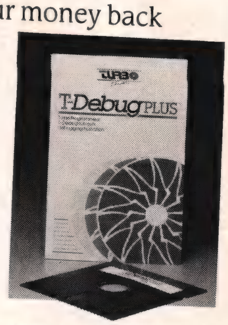
"Impressive products...I recommend them."

Philippe Kahn, President
Borland International

Satisfaction guaranteed or your money back within 30 days.

Call toll free for credit card orders:

800-538-8157 x830
outside California or
800-672-3470 x 830
in California



Shipping and taxes prepaid for U.S. and Canadian customers. Others please include $6 per item for shipping.

TurboPower Software products require Turbo Pascal 3.0 (standard, 8087, or BCD) and require PC-DOS 2.X or 3.X, and run on the IBM PC/XT/AT and compatibles.

TURBO Power

TurboPower Software, 3109 Scotts Valley Dr. #122
Scotts Valley, CA 95066
Call 408-438-8608 for more information (M-F 9AM-5PM PST).

Circle **no. 207** on reader service card.

(continued from page 20)

neither a *LEAF* nor a *BLEND*, it is marked as a *WASH*. This happens when none of the four children share a common color either because they are really four different colors or some of them are *WASH* nodes and have no color information. The node type is used later when this node's parent node is passed to *condense()* and this node is then one of the child nodes. It is also used during output of the quadtree because *WASH* nodes are not output.

Relnode() is the complement of *crtnode()*—it releases an unneeded node. For efficiency, this routine could add the node to a linked list of free nodes from which *crtnode()* could get space for new nodes. It is implemented here as a call to the system routine *free()*.

Getcolor(), called by *addnode()* when it reaches a leaf node at the bottom level, is a function to return the color of the pixel represented by a bottom-level node. The majority of the code is concerned with converting the position of the node in the quadtree, given as its locational code, to a column and row (x and y) pixel position. The code does this by extracting the directions shifted into the locational code by *addnode()* in the process of building the locational code. Because this is a bottom-level node, the number of direction values is equal to the top-level number. *Getcolor()* shifts the direction value for each level, starting with the top level or root, to the bottom bits and masks them off. As the direction code is recovered for each level, the column and row values are shifted left one bit and the new bottom bit is set or not depending on the direction. This works because the direction values define two simultaneous binary searches through the pixel space. The first search, for column, successively splits the pixel space into left and right halves. The second search, for row, successively splits the pixel space into upper and lower halves. Thus each bit in the column or row value is a direction in the binary search. Whereas the locational code needs two bits to represent one of four directions, the column and row values need one bit to represent one of two directions.

Outtree() and *outnode()* are the control routines for the second phase of Listing One—outputting the quadtree as locational codes. In order to output the nodes in a breadth-first order—that is, all nodes at one level are output before any lower-level nodes—*outtree()* and *outnode()* construct a linked list of nodes yet to be dealt with. As the node at the front of the list is examined, and possibly output, its children are added to the end of the list. The linked list serves as a FIFO queue, which means that each level, starting from the top, is processed before the next level is started. *Outtree()* initializes the linked list by putting the root node on it and setting its next pointer to *NULL*. It then enters a loop calling *outnode()* with the next node on the list until the list is exhausted. *Outnode()* checks the node type of the current node and outputs the locational code and color if the node is not a *WASH*. During this final pruning of the quadtree, nodes whose only function in the quadtree was to point to lower-level nodes are dropped from the locational code form because the pointers are no longer needed. The routine *putlcc()* is not defined here because what it does could be system or application dependent. The simplest thing for *putlcc()* to do is write the data to a file for later processing or display. *Putlcc()* could also transmit the data to a remote display. After *putlcc()* is called, the node's children, if any, are added to the end of the list.

Listing Two

Listing Two, page 44, presents a set of routines for displaying a quadtree from a locational code form. As with the first set of routines, no main routine is given and only a single routine needs to be called. The main routine will have to do whatever initialization is necessary, such as opening the file containing the quadtree and reading in a header section or opening a communications channel. It may also need to initialize a graphics system or at least clear the screen. *Qdisp()* is the entry point for the second set of routines. It needs to know the size of the original image, which could be read from a file header associated with the quadtree file or supplied by the user. These routines make calls to two externally defined routines that you must supply. These are *getnxn()*, which returns the next quadtree node as a locational code and a color, and *filrec()*, which fills in a rectangle on the display with a given color.

Most of *qdisp()* is a loop that gets the next node as a locational code and color by a call to *getnxn()*, converts the locational code to the corner and side of the square represented, and fills in the square with the color by a call to *filrec()*. *Getnxn()* and *filrec()* are assumed to be supplied separately because they might be both system and application dependent. *Getnxn()*, for instance, might be reading the quadtree data from a file or reading it from a serial port. *Filrec()* is given the upper-left corner and sides of a rectangle (node quadrants are, of course, always square, but *filrec()* is presumed to be more general) and a color, and it fills the defined rectangle on the display with the color. If you are lucky enough to be using a system with a graphics package that supplies such a call (or better yet, a display that has the function available in hardware), then the implementation of *filrec()* should be simple. If you are not so lucky, then *filrec()* may have to loop over all the pixels in the rectangle, setting each to the given color.

The routine *square()* converts a locational code to the corner and side of the square represented by the node. It is very similar to *getcolor()* in Listing One. Because the level of the node is not known, the code must search the locational code for the beginning marker. After finding this, the code loops, like *getcolor()*, over the direction values from the root to the current node. At each iteration, the length of the side, initialized to the original image size, is divided by two and the corner position is adjusted according to which quadrant is indicated.

Practical Considerations

Quadtrees in pointer form can use up a lot of memory. In the worst case, in which no nodes can be released during construction, an image of size N $(= 2^k)$ would require more than $N \times N$ nodes. For example, an image of 256×256 pixels could require more than 64K nodes. Using my data structure for a node, this takes up 2 megabytes of memory on a machine that uses 4 bytes for *ints* and pointers. A more efficient structure may be needed. Using

smaller fields and/or combining fields would be one way to reduce memory needs. The node type field could be removed with some additional processing—the type *LEAF* could be deduced from the fact that all child pointers are *NULL* and, because the color of a *WASH* node is meaningless, a special value in the color field could indicate that a node was a *WASH*.

Besides the memory problem, the time required to create the quadtree may also prove to be a problem. Despite some efforts to speed up the process, such as switching to a base 4 representation for the locational codes, creating the quadtree is still very slow. One improvement, as mentioned already, might be to change the way *crtnode()* and *relnode()* get and release nodes. Another might be to keep more information about the location of the current node so that *getcolor()* does not have to figure this out from the locational codes. I think, however, that major improvements will require somehow avoiding all the hundreds (or thousands) of calls to *addnode()* and *condense()* for sections of the image that are a single color and could be quickly defined as a few high-level nodes.

Much to my disappointment, the display of the quadtree is not very fast. Even on a display that supports rectangle fill, a scan-line display of an image is faster than the quadtree display, although the display of the quadtree is more interesting to watch. For any reasonably interesting image, a lot of individual pixels must ultimately be filled in to complete the image and this takes a lot of time.

Despite these problems, quadtrees can offer some advantages over other graphical image representations and, in some cases, may be the best choice.

Availability

All the source code for articles in this issue (except C Chest) is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and disk format (MS-DOS, Macintosh, Kaypro).

Bibliography

Gargantini, I. "An Effective Way to Represent Quadtrees." *Communications of the ACM*, vol. 25, no.12 (December 1982): 905–910.

Jones, L., and Iyengar, S. "Representation of Regions as a Forest of Quadtrees." *Proceedings of the IEEE Conference on Pattern Recognition* (1981): 57–59.

Samet, H. "An Algorithm for Converting Rasters to Quadtrees." *IEEE Transactions in Pattern Analysis and Machine Intelligence*, vol. 3, no. 1 (January 1981): 93–95.

Samet, H. "Data Structures for Quadtree Approximation and Compression." *Communications of the ACM*, vol. 28, no. 9 (September 1985): 973–993.

Scott, S., and Iyengar, S. "TID—A Translation Invariant Data Structure for Storing Images." *Communications of the ACM*, vol.29, no. 5 (May 1986): 418–429.

Witten, I. H., and Cleary, J. G. "Foretelling the Future by Adaptive Modeling." *ABACUS*, vol.3, no.3 (Spring 1986): 16–36, 73.

(Listings begin on page 40.)

Vote for your favorite feature/article.
Circle Reader Service **No. 1**.

ARC Wars: MS-DOS Archiving Utilities

by Russell Nelson

Since the dawn of computers, people have been trying to make their computers run faster. One speedup technique is data compression, which lets the computer operate on less data while still accomplishing the same amount of work.

Data compression relies on the fact that most data are not random. English language text, for example, has a known character distribution—the letter *E* occurs more often than *T*, *T* more often than *O*, and so on. A data compression algorithm relies on these quirks to use fewer bits to represent the same data. For users downloading from bulletin boards, this approach translates into lower phone bills.

The old standard for data compression was a combination of three programs—one to combine files into one library (LU, library utility), one to squeeze this library into fewer bits (SQ), and another to unsqueeze the library (USQ). The squeeze program would produce a Huffman encoding[1] of its input file.

Thom Henderson of System Enhancement Associates (SEA) created ARC to provide an alternative to LU. ARC can add files to an archive and automatically determine which of four different compression methods to use. An archive can never be much larger than the component files because one of the four "compression" methods consists simply of storing

Data compression relies on the fact that most data are not random.

the file unaltered.

In actual use, the savings are often considerable. After automatic compression was added, ARC performed better than the LU/SQ/USQ combination, and within seven months of its introduction, it became the new standard for BBS files. This could have happened just because it was more convenient, but more probably it was because it used an improved compression algorithm—the Lempel-Ziv[2] algorithm.

In fact, ARC became so popular that spin-offs using the same file format appeared. Spin-offs were easy to develop because SEA made the ARC source available. At present, four different ARC-compatible systems and one ARC-incompatible system are in use. This article reviews the performance of each of them on different sets of files.

The Programs

Table 1, below, lists the current (as of October 1986) program versions that comprise each of the five archiving systems. Note that two of the systems are distributed as separate programs—PKARC and PKXARC form one system; ARCA, ARCE, and ARCV form another. The other three systems are ARC, ARCH, and ZOO.

The Donation column in Table 1 gives the suggested donation if the software is shareware. Some of the software is copyrighted, but no donation is suggested. The Cost column gives the cost of the software including automatic updates, printed documentation, and so on. All these programs are freely copyable.

The box on page 28 gives the author's address for each system. In addition, all the files mentioned in this article are available from the Clarkson University Heath Users Group (CUHUG) Fido—(315) 268-6667, 300/1,200/2,400 baud, 24 hours—as well as from many other BBSs. ARC and PKXARC are distributed as self-extracting .COM files; the other programs are distributed in archive form. Look for ARC*.* and PKX*.* if the BBS has a wildcard list function.

Program	Version	Size (bytes)	Donation	Cost
ARC	5.12	32429	$35	$50
ARCA	1.18	3796	none	
ARCE	2.06	5424	none	
ARCV	1.15	2063	none	
ARCH	5.38	32694	none	
PKARC	1.1	15972	$15	$35
PKXARC	3.2	9984	$15	$35
ZOO	1.20	29120	none	

Table 1: *Program versions*

Russell Nelson, 11 Grant St., Potsdam, NY 13676. Russell is the author of Painter's Apprentice. He holds an M.S.E.E. degree from Clarkson University in Potsdam, New York.

In addition to the archive programs discussed in this article, the CUHUG Fido has the following related files: DARC.ARC, which deletes from the disk any files that may be found in the archive; XONE2.ARC, which extracts one file from an archive into a new archive containing just that file; LZ.ARC, which contains assembly-language source for a Lempel-Ziv compresser and decompresser; ARCX.ARC, which contains Turbo Pascal source for an archive extractor; and ARC44.ARC, which contains the source for Version 4.4 of ARC.

Comparison of Features

Table 2, below, lists the features that each program provides. Most of the titles are self-explanatory; those that aren't are:

- Add files to archive: Obviously all archive systems can add files to an archive but not all programs in an archive system can do so.
- Alphabetic file names: Because ARC uses a distributed directory (the file names are not kept in a central location), alphabetic adding means that the files in the archive must be reordered when a new file is added and a copy of the archive must be made. The advantage of not copying the archive is that an archive can fill a whole floppy rather than being limited to just half. ZOO uses a version numbering scheme to avoid copying. ARCA simply ignores the issue and makes two copies of the file, only the first of which is accessible.
- Damaged headers: If an archive gets munged, a file header can be damaged. Some archivers can skip the damaged file; some just give up.
- Extract to explicit path: Sometimes you might want to extract a file to a subdirectory/drive other than the current one.
- Forced storing: Because data compression can take a fair bit of time, some archivers allow you to force the files to be added uncompressed for later compression of the entire archive.
- Freshen files already in archive: Only newer files already in the archive are archived.
- List file names only: This is useful if you want to pipe the list of file names to another program.
- Only packing and crunching: Some archivers don't bother to squeeze or store a file.
- Update adds only newer files: Older files in the archive are left alone.
- Wildcard archive file names: You can specify an ambiguous archive file name using wildcards in combination with some operations.

Benchmarks

I ran benchmarks using a 5-MHz Z100 with a V20 to run MS-DOS 2.18—all the programs run under generic MS-DOS. I stored the files and programs in a RAM disk so that physical disk access times were not significant. The verbose listings were redirected to a file, so console output time is not reflected in the run time.

I have tried to use test data that is easily obtainable. ARC44 is an archive of the source of ARC, Version 4.4, and I used it to test the ability of an archiver to cope with a file that cannot be compressed further. The MASM benchmark is the MASM, Version 4.0, distribution disk, and I included it to test the compression of nontext files. The TDebug benchmark is the source of TDebug Plus, available from Turbo Power Software; I included it to test the compression of text files.

I tested only the most common operations—add, add to existing, delete, list, and extract. Results of the benchmark tests are shown in Table 3, page 28. I assume less common operations, such as update, freshen, and move, will be roughly similar in speed. Example 1, page 30, shows the output of verbose listings for some of the archive systems. There is not much to say because all give the same information and have similar run times.

	ARC	ARCA	ARCE	ARCV	ARCH	PKARC	PKXARC	ZOO
Add files to archive	y	y	-	-	y	y	-	y
Alphabetic file names	y	-	-	-	y	y	-	-
Archive file compatible	y	y	y	y	y	y	y	-
Comments attached to files	-	-	-	-	-	y	-	y
Copies while modifying	y	-	-	-	y	y	-	-
Damaged headers	y	y	-	-	y	-	-	n/a
Delete files after adding	y	y	-	-	y	y	-	y
Encryption	y	-	-	-	-	-	-	-
Extract files to console	y	-	-	-	y	-	y	y
Extract files to printer	-	-	-	-	-	-	y	-
Extract into archive	-	-	-	-	y	-	-	-
Extract to explicit path	-	-	y	-	-	-	y	-
Forced storing	y	-	-	-	y	-	-	y
Freshen files already in archive	y	-	-	-	y	y	-	y
List file names only	-	-	-	-	y	-	-	y
Make backup of the archive file	y	-	-	-	y	-	-	y
Only packing and crunching	-	y	-	-	-	-	-	-
Replace existing files on extract	y	-	y	-	y	-	y	y
Test archive integrity	y	-	-	-	y	-	y	y
Update adds only newer files	y	-	-	-	y	y	-	y
Verbose listing of archive	y	-	-	-	y	-	y	y
Wildcard archive file names	-	-	y	y	y	y	y	y

Table 2: *Comparison of features*

ARC WARS
(continued from page 27)

Conclusions

As shown in Table 3, PKARC is the fastest archiver by a wide margin, and PKXARC is the fastest extractor. PKARC also produced the smallest archives in all but one instance, in which ZOO was slightly smaller. ARCA/ARCE/ARCV, PKARC/PKXARC, and ZOO are written in assembly language, whereas ARC and ARCH are written in C. If you don't mind the donation, PKARC/PKXARC is the system to use.

ZOO performed adequately. ZOO is the only explicitly public-domain ar-

Authors/Vendors

ARC
Thom Henderson
System Enhancement Associates
21 New St.
Wayne, NJ 07470
For $50 you receive a program disk with printed documentation. If you obtain ARC by other means, then you cannot use it in a commercial environment or a government organization unless you pay a $35 license fee. Site licenses and commercial distribution licenses are available, as is the full program source.

ARCA, ARCE, ARCV
Vernon D. Buerg
456 Lakeshire Dr.
Daly City, CA 94015
CompuServe: 70007,1212
Data/RBBS: (415) 994-2944

ARCH
Les Satenstein
PCOM RBBS Montreal
(514) 989-9450
Given the similarity in features, run time, and results, ARCH must be a modified copy of ARC. The ARC copyright permissions strictly prohibit distribution of modified copies of ARC. Nevertheless, ARCH has more features than ARC, and so I included it in this review.

PKARC, PKXARC
Phil Katz
7032 Ardara Ave.
Glendale, WI 53209
Send comments to:
Exec-PC multiuser IBM BBS
modem: (414) 964-5160
If you find PKARC and PKXARC fast, easy, and convenient to use, a contribution of $15 would be appreciated. With each contribution of $35 or more, you receive free upgrades of the next versions of PKARC and PKXARC when available, including documentation.

ZOO
Rahul Dhesi
GEnie: DHESI
People/Link: OLS806
ARPAnet/CSnet: dhesi%bsu@csnet-relay.ARPA
UUCP: !seismo!csnet relay .ARPA!bsu!dhesi
ZOO is in the public domain.

Archive add—ARC44—archive of source of ARC (57,728 bytes)

	ARC	ARCA	ARCH	PKARC	ZOO
Run time (seconds)	144.66	42.42	150.27	25.86	40.41
Size (bytes)	57,728	58,014	57,728	57,728	57,728
Total size (bytes)	57,759	65,564	57,759	57,759	60,625
Stowage	stored	packed	stored	stored	stored

Archive add—MASM 4.00 distribution disk (288,122 bytes total)

	ARC	ARCA	ARCH	PKARC	ZOO
Run time (seconds)	648.77	122.41	653.73	92.11	124.91
Total size (bytes)	237,072	221,751	237,072	221,020	221,907
Compression (percent)	17	23	17	23	23

Archive add—TDebug Plus source (289,049 bytes total, all ASCII)

	ARC	ARCA	ARCH	PKARC	ZOO
Run time (seconds)	373.44	86.56	364.17	65.36	82.25
Total size (bytes)	116,802	116,255	116,802	115,950	113,013
Compression (percent)	59	59	59	59	61

Archive extract—ARC44—archive of source of ARC (57,728 bytes)

	ARC	ARCE	ARCH	PKXARC	ZOO
Run time	136.87	51.27	139.75	50.98	n/a

Archive extract—MASM 4.00 distribution disk (288,122 bytes total, 13,595 bytes ASCII)

	ARC	ARCE	ARCH	PKXARC	ZOO
Run time	244.39	58.06	252.00	46.74	69.64

Archive extract—TDebug Plus source (289,049 bytes total, all ASCII)

	ARC	ARCE	ARCH	PKXARC	ZOO
Run time	182.60	42.73	188.30	33.43	54.86

***Table 3:** Benchmark results*

Techical Product Information.

Dr. Dobb's Journal of Software Tools

Name ______________________________

Title ______________________________

Company ____________________ Phone ____________________

Address ______________________________

City/State/Zip ______________________________

March 1987 #125 **Expiration Date: June 30, 1987**

Please circle one letter in each category:

I. My work is performed:
A. for in-house use only.
B. for other companies.
C. for end users/retailers.
D. in none of the above areas.

II. My primary job function:
A. Software Project Mgmt/Spvr
B. Hardware Project Mgmt/Spvr
C. Computer Consultant
D. Corporate Consultant
E. Other

III. My company department performs:
A. software development.
B. computer system integration.
C. computer manufacturing.
D. computer consulting.
E. computer research
F. none of the above.

IV. This inquiry is for:
A. a purchase within 1 month.
B. a purchase within 1 to 6 months.
C. product information only.

V. Corporate Purchase Authority:
A. Final Decision-maker
B. Approve/Recommend
C. No Influence

VI. Personal Computer Users at my Jobsite:
A. 10,000 or more
B. 500 to 9,999
C. 100 to 499
D. 10 to 99
E. less than 10

VII. On average, I advise others about computers:
A. more than once per day.
B. once per day.
C. once per week.
D. less than once per week.

VIII. In my job function, I:
A. design software and/or write code.
B. design software.
C. write code.
D. don't design software or write code.

A Reader Service number appears on each advertisement. Circle the corresponding numbers at right for more info.

1	2	3	4	5	6	7	8	9
11	12	13	14	15	16	17	18	19
21	22	23	24	25	26	27	28	29
31	32	33	34	35	36	37	38	39
41	42	43	44	45	46	47	48	49
51	52	53	54	55	56	57	58	59
61	62	63	64	65	66	67	68	69
71	72	73	74	75	76	77	78	79
81	82	83	84	85	86	87	88	89
91	92	93	94	95	96	97	98	99
101	102	103	104	105	106	107	108	109
111	112	113	114	115	116	117	118	119
121	122	123	124	125	126	127	128	129
131	132	133	134	135	136	137	138	139
141	142	143	144	145	146	147	148	149
151	152	153	154	155	156	157	158	159
161	162	163	164	165	166	167	168	169
171	172	173	174	175	176	177	178	179
181	182	183	184	185	186	187	188	189
191	192	193	194	195	196	197	198	199
201	202	203	204	205	206	207	208	209
211	212	213	214	215	216	217	218	219
221	222	223	224	225	226	227	228	229
231	232	233	234	235	236	237	238	239
241	242	243	244	245	246	247	248	249
251	252	253	254	255	256	257	258	259
261	262	263	264	265	266	267	268	269
271	272	273	274	275	276	277	278	279
281	282	283	284	285	286	287	288	289
291	292	293	294	295	296	297	298	299
301	302	303	304	305	306	307	308	309
311	312	313	314	315	316	317	318	319
321	322	323	324	325	326	327	328	329
331	332	333	334	335	336	337	338	339
341	342	343	344	345	346	347	348	349
351	352	353	354	355	356	357	358	359
361	362	363	364	365	366	367	368	369
371	372	373	374	375	376	377	378	379
381	382	383	384	385	386	387	388	389
391	392	393	394	395	396	397	398	399

Circle 999 to start a 12 month subscript at the price of $29.97

TAKE THIS CARD WITH YOU AS YOU READ THROUGH THIS ISSUE OF DR. DOBB'S.

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER
P.O. Box 2157
Clinton, Iowa 52735-2157

Technical Product Information.

TAKE THIS CARD WITH YOU AS YOU READ THROUGH THIS ISSUE OF DR. DOBB'S.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
01	102	103	104	105	106	107	108	109	110
11	112	113	114	115	116	117	118	119	120
21	122	123	124	125	126	127	128	129	130
31	132	133	134	135	136	137	138	139	140
41	142	143	144	145	146	147	148	149	150
51	152	153	154	155	156	157	158	159	160
61	162	163	164	165	166	167	168	169	170
71	172	173	174	175	176	177	178	179	180
81	182	183	184	185	186	187	188	189	190
91	192	193	194	195	196	197	198	199	200
01	202	203	204	205	206	207	208	209	210
11	212	213	214	215	216	217	218	219	220
21	222	223	224	225	226	227	228	229	230
31	232	233	234	235	236	237	238	239	240
41	242	243	244	245	246	247	248	249	250
51	252	253	254	255	256	257	258	259	260
61	262	263	264	265	266	267	268	269	270
71	272	273	274	275	276	277	278	279	280
81	282	283	284	285	286	287	288	289	290
91	292	293	294	295	296	297	298	299	300
01	302	303	304	305	306	307	308	309	310
11	312	313	314	315	316	317	318	319	320
21	322	323	324	325	326	327	328	329	330
31	332	333	334	335	336	337	338	339	340
1	342	343	344	345	346	347	348	349	350
1	352	353	354	355	356	357	358	359	360
1	362	363	364	365	366	367	368	369	370
1	372	373	374	375	376	377	378	379	380
1	382	383	384	385	386	387	388	389	390
1	392	393	394	395	396	397	398	399	999

rcle 999 to start a 12 month subscription
: the price of $29.97

Dr. Dobb's Journal of Software Tools

Name ______________________________
Title ______________________________
Company ______________ Phone ______________
Address ______________________________
City/State/Zip ______________________________

March 1987 #125 **Expiration Date: June 30, 1987**

Please circle one letter in each category:

I. My work is performed:
A. for in-house use only.
B. for other companies.
C. for end users/retailers.
D. in none of the above areas.

II. My primary job function:
A. Software Project Mgmt/Spvr
B. Hardware Project Mgmt/Spvr
C. Computer Consultant
D. Corporate Consultant
E. Other

III. My company department performs:
A. software development.
B. computer system integration.
C. computer manufacturing.
D. computer consulting.
E. computer research
F. none of the above.

IV. This inquiry is for:
A. a purchase within 1 month.
B. a purchase within 1 to 6 months.
C. product information only.

V. Corporate Purchase Authority:
A. Final Decision-maker
B. Approve/Recommend
C. No Influence

VI. Personal Computer Users at my Jobsite:
A. 10,000 or more
B. 500 to 9,999
C. 100 to 499
D. 10 to 99
E. less than 10

VII. On average, I advise others about computers:
A. more than once per day.
B. once per day.
C. once per week.
D. less than once per week.

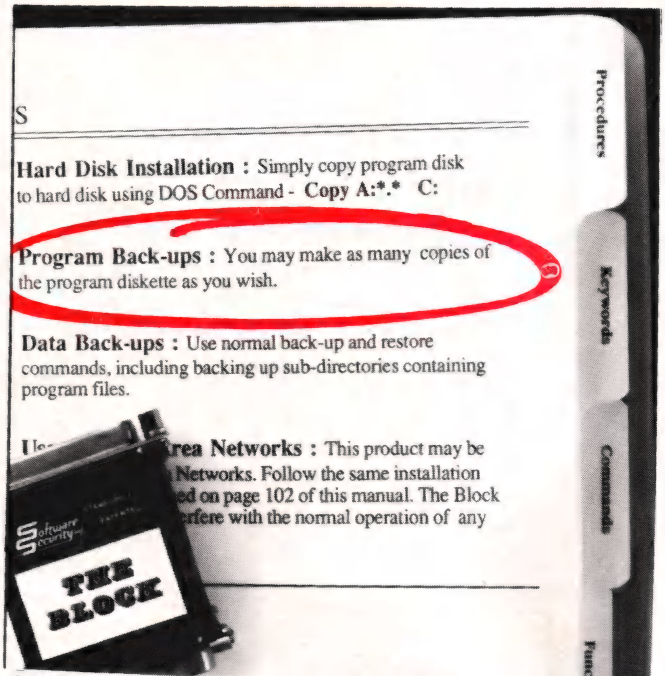
VIII. In my job function, I:
A. design software and/or write code.
B. design software.
C. write code.
D. don't design software or write code.

A Reader Service number appears on each advertisement. Circle the corresponding numbers at left for more info.

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this: *Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.*

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905
203 329 8870

Circle **no. 170** on reader service card.

ARC WARS
(continued from page 28)

chiver I reviewed, which is its strongest point. Its weakest point is that its archive files (.ZOO) are not compatible with ARC-type archive files (.ARC). ZOO's author plans to put the finished source in the public domain, so I expect someone will convert it to use ARC-type files.

Surprisingly, not one of the archive programs includes a "rename file in archive" command. In addition, both ARC and PKARC are distributed in a self-extracting .COM file but provide no facility for creating your own self-extracting .COM file. Another nice member of the ARC family would be a "ROM" disk driver that takes an archive file as input and produces a read-only disk drive when installed. A ROM disk would be handy for frequently used files.

Notes

1. D. Huffman, "A Method for Constructing Minimum Redundancy Codes," *Proceedings of the Institute of Radio Engineers,* vol. 40 (May 1952): 1098–1101.

2. Terry A. Welch, "A Technique for High Performance Data Compression." *IEEE Computer,* vol. 17, no. 6 (June, 1984).


```
---------------arc-----------------
Name        Length      Stowage     SF          Size now    Date        Time        CRC
-----------------------------------------------------------------------------------------
ARC.H         1841      crunched    39%           1133      9 Nov 85    10:25p      F24F
ARC.M          512      packed      34%            343     22 Aug 85    12:18a      3057
LOAD.BAT      1024      crunched    49%            531     12 Nov 85    11:02a      725A
XARC.MAC      3584      crunched    41%           2142      8 Nov 85    11:37p      6B57
-------------------------------------------------------
Total     22 117610                 52%          57063
Run time: 319

---------------arch----------------
Doing ARC44.ARC:
Name        Length      Stowage     SF          Size now    Date        Time
-----------------------------------------------------------------------------------------
ARC.H         1841      crunched    39%           1133      9 Nov 85    10:25p
ARC.M          512      packed      34%            343     22 Aug 85    12:18a
LOAD.BAT      1024      crunched    49%            531     12 Nov 85    11:02a
XARC.MAC      3584      crunched    41%           2142      8 Nov 85    11:37p
--------------------
   22 files 117610
Run time: 200

---------------arcv----------------
Archive: ARC44.ARC
Name        Length      Stowage     SF          Size now    Date        Time        CRC
-----------------------------------------------------------------------------------------
ARC.H         1841      crunched    38%           1133      09 Nov 85   22:25       F24F
ARC.M          512      packed      33%            343      22 Aug 85   12:18       3057
LOAD.BAT      1024      crunched    48%            531      12 Nov 85   11:02       725A
XARC.MAC      3584      crunched    40%           2142      08 Nov 85   23:37       6B57
-------------------------------------------------------
*total    22 117610                 51%          57063
Run time: 164

---------------pkxarc--------------
PKXARC         FAST!       Archive Extract Utility           Version 3.2        9-12-86
Copyright (c) 1986 Phil Katz, All Rights Reserved. PKXARC/h for help
Searching: ARC44.ARC
Name        Length      Method      Size        Ratio       Date        Time
-----------------------------------------------------------------------------------------
ARC.H         1841      crunched     1133       39%         11-09-85    22:25:16
ARC.M          512      packed        343       34%         08-22-85    12:18:44
LOAD.BAT      1024      crunched      531       49%         11-12-85    11:02:58
XARC.MAC      3584      crunched     2142       41%         11-08-85    23:37:48
-------------------------------------------------------
0022          117610                57063       52%
Run time: 786
```

Example 1: *Output of verbose listings*

CODE BLUE

PANASONIC FX-800
80286 8MHz • 1.2 MB Floppy
30 MB Hard Disk • 512K RAM
Monitor and Graphics Card **$2199**
Custom configurations available

PANASONIC FX-800
8086 7.14 MHz • One 360K Floppy
20 MB Hard Disk • 640K
Monitor and Graphics Card **$1375**

TOSHIBA T1100 PLUS
2 720KB Drives
8 hr. Battery • 640K **CALL**

SOFTWARE POTPOURRI

Aldebaran Labs	
Source Print	109
Compact Source Print	CALL
Tree Diagrammer	CALL
Alpha Computer	
ACS Time Series	399
For-Winds	75
Forlib-Plus	52
Scientific Subroutine Pkg.	254
Strings & Things	52
Please specify Fortran Compiler	
Arity	
Expert System Dev. Pkg.	269
File Interchange Toolkit	45
Prolog Compiler/Interpreter	729
Prolog Interpreter	319
Screen Design Toolkit	45
SQL Development Package	269
Standard Prolog	78
Combo Package	1099
Blaise	
Asynch Manager–C	131
Asynch Manager–Pascal	131
C Tools	98
C Tools 2	78
C Tools Plus	134
EXEC Program Chainer	74
Pascal Tools	98
Pascal Tools 2	78
Pascal Tools & Tools 2	134
Runoff Text Formatter	44
Turbo Asynch Plus	78
Turbo Power Tools Plus	78
View Manager–C	195
View Manager–Pascal	195
Borland	
Reflex	97
Reflex Workshop	47
Reflex & Reflex Workshop	141
Turbo Database Toolbox	47
Turbo Editor Toolbox	47
Turbo Gameworks Toolbox	47
Turbo Graphix Toolbox	47
Word Wizard	47
Turbo Lightning	65
Turbo Pascal w/8087 & BCD	65
Turbo Prolog	64
Turbo Tutor for Pascal	27
Word Wizard & Turbo Lightning	98
Turbo Pascal–Macintosh	71
Computer Innovations	
C-86	279
Introducing C	99
C to dBase	129
CI-86 Plus	CALL
CI Probe	189
CI ROMPac	139
Entelekon	
Combo	165
C Function Library	105
C Windows	105
Superfonts for C	42
Essential Software	
C Essentials	82
C Utility Library	135
Essential Graphics	189
GSS	
Graphics Development Toolkit	369
Kernel System (DOS)	369
Kernel System (IBM RT)	639
Metafile Interpreter	229
Plotting System	374
Solutions Chart	235
Solutions Plottalk	235
Solutions Terminal	235
Intel	
386 Board (Avail. April)	CALL
Laboratory Micro	
PC/Forth	114
PC/Forth Plus	204
Adv. Color Graphics Support	75
Enhanced Graphics Support	154
8087 Support	75
Interactive Symbolic Debugger	75
Native Code Optimizer	154
PCTERM	75
Software Floating Point	75
Lattice	
C Compiler	289
C Compiler w/Library S.C.	544
C XREF Generator	38
C XREF Generator w/S.C.	147
C-Sprite	135
Curses Screen Mgr.	93
Curses Screen Mgr. w/S.C.	181
dBC	186
dBC w/Source Code	369
LMK Make Facility	144
RPGII Combo	899
RPGII Compiler–No Royalty	631
Secret Disk	92
SideTalk	92
Text Management Utilities	92
TopView Toolbasket	186
TopView Toolbasket w/S.C.	375
Z-80 C Cross Compiler	375
Z-80 C X-Compiler w/S.C.	744
Logitech	
Modula-2/86 Compiler	63
Modula 2/86 w/8087	99
Modula 2/86 PLUS	144
Modula 2 Library Sources	84
Modula 2 Make Utility	26
Modula 2 ROM Package	177
Modula 2 RunTime Debug	57
Turbo-Modula Translator	43
Modula 2 Utilities Pack.	43
Modula 2 Windows Pack.	43
Mice listed on next page.	
MetaGraphics	
MetaWindows	135
MetaWindows Plus	195
TurboWindows	65
MicroFocus	
All products available.	
MicroHelp	
Peeks n Pokes	38
Inside Track	52
MACH 2	62
Stay-Res	79
MicroSoft	
QuickBasic 2.0	65
Basic Interpreter (XENIX)	215
C Compiler	282
Cobol Compiler	435
Cobol Compiler (XENIX)	619
Cobol Tools	199
Cobol Tools (XENIX)	299
Fortran Compiler	207
Fortran Compiler (XENIX)	559
LISP	159
Macro Assembler	94
Bus Mouse	119
Serial Mouse	129
Sort	134
muMath & muSimp	189
Pascal Compiler	184
Pascal Compiler (XENIX)	429
Tech. Ref. Encylopedia	99
Windows	65
Windows Development Kit	324
Phoenix	
Pasm86 Macro Assembler	129
Pdisk Hard Disk Utility	129
Pfantasy Pac	CALL
Pfinish Performance Anal.	235
Pfix-86 + Symbolic Debug	235
PforCe C Library	235
Plink-86 + Enhanced Linker	319
Pmaker Make Utility	79
Pmate Macro Text Editor	119
Pre-C Lint Utility	159
Ptel Binary File Transfer	115
Polytron	
PolyBoost	65
C Beautifier	43
C Library I	73
Power Comm.	134
PolyLibrarian	74
PolyLibrarian II	110
PolyMake	74
PolyOverlay	74
PolyXREF–Complete	175
PolyXREF–One Language	105
PVCS Version Control Sys.	319
PVMFM Virtual Mem Mgr.	144
Raima	
All products available.	
RDS	
All products available.	
Relia	
Cobol	829
Ryan-McFarland	
RM/Cobol (XENIX)	975
RM/Fortran (XENIX)	589
RM/Cobol	615
RM/Cobol 8X ANSI 85	875
RM/Fortran	379
Santa Cruz Operation	
Complete XENIX System	1049
XENIX Development System	519
XENIX Operating System	519
XENIX Test Processing Package	149
Lyrix	479
Networks for XENIX	519
SCO Professional	685
Softcraft	
Btrieve ISAM Manager	189
Xtrieve Query Utility	162
Rtrieve Report Generator	119
Btrieve/N Networks	459
Xtrieve/N Networks	CALL
Rtrieve/N Networks	CALL
Software Bottling	
Flash-Up Windows	75
Flash Code	119
Screen Sculptor	94
Speed Screen	28
STSC	
APL PLUS/PC	439
APL PLUS/PC Spreadsheet	149
APL PLUS/PC Tools Vol. 1	229
APL PLUS/PC Tools Vol. 2	59
APL PLUS/UNX (XENIX)	695
Financial/Stat. Library	194
Pocket APL	75
StatGraphics	589
Summit Software	
BetterBasic	139
BetterBasic 8087 Support	74
BetterBasic Btrieve Interface	74
BetterBasic C Interface	CALL
BetterBasic RunTime Module	219
Toshiba	
3.5" Drive Kit	139
True Basic	
True Basic w/Converter	105
True Basic w/Converter/RunTime	189
Advanced String Library	42
Asynch Communication Support	42
BasicA Converter	42
Btrieve Interface	42
Developer's Toolkit	42
Formlib	42
Hercules Graphics Support	42
Sorting & Searching	42
RunTime Module	99
TurboPower Software	
T-Debug	52
Turbo Extender	65
Turbo Power Utilities w/source	79
Turbo Power Utilities no source	49
Wendin	
Operating System Toolbox	77
PCNX Operating System	77
PCVMS	77
XTC Text Editor with Source	77

Optimizing Integer Multiplications by Constant Multipliers

by Robert D. Grappel

> ***Some multiplications can be sped up by 'unrolling' the calculation.***

Integer multiplication by a constant multiplier can occur frequently in high-level language programs. Besides the explicitly coded multiplications, the compiler must generate multiplication instructions as part of each array reference. To address an array element, the compiler forms code to multiply the array index by the size of an array element (a constant). If the array elements are simple data types (bytes, words, and so on) then the multiplication is often done as a shift left because the size of the element is a power of 2. Some of the more powerful processors (Intel 80386, Motorola 68020, National 32016, and so on) provide scaled-indexed addressing modes that incorporate the appropriate shift as part of the address calculation.

If the array element is not a simple type, however, the multiplication must be done explicitly. Multiply dimensioned arrays require a multiplication for each index (except, perhaps, for the last one, which can be done with a shift).

Multiplication is a time-consuming operation, even on those processors that have multiplication instructions. Table 1, right, shows the execution times, in clock cycles, for several types of instructions, including multiplication, on some modern microprocessors. The 68000, for example, requires about 70 clocks for a 16-bit register-to-register multiplication instruction, compared to about 8 clocks for a 32-bit register-to-register addition or subtraction. A 32-bit register shift requires about 6 clocks plus an additional 2 clocks per shift position. Clearly, the 68000 can do several adds, subtracts, and shifts in the time it takes to perform one multiplication.

Indexing an array (without artificially limiting the size to 64K) requires a 32-bit multiplication, which neccessitates at least two 16-bit multiplications and an addition on the 68000 along with some logic. Because this 32-bit multiplication is likely to be done as a run-time subroutine, there is often an additional setup and calling overhead, too. (The 16-bit multiplication of the Intel 80286 is sufficient to address an entire memory segment.) Thus, there is room for a compiler to fabricate an optimized sequence of additions, subtractions, and shifts in place of a multiplication on any of these processors.

Robert D. Grappel, 28 Buckmaster Dr., Concord, MA 01742. Robert Grappel has a Ph.D. in solid-state physics from Ohio University. He is currently a consultant involved in the design of new air-traffic control systems.

Unrolling the Loop

Computers multiply numbers using some variation of the following algorithm:

1. Clear work register *Rw*, which becomes the product.
2. If the low-order bit of the multiplier is a 1, add the multiplicand to *Rw*.
3. Shift the multiplier right one bit position.
4. If the multiplier is 0, stop (product in *Rw*).
5. Shift the multiplicand left one bit position.
6. Go to step 2.

It is apparent that the computer performs multiplication as a sequence of shifts and additions—step 2 is an addition; steps 3 and 5 are shifts. If the multiplier is a constant, the algorithm can be "unrolled" into a sequence that includes only adds and shifts. This sequence is called a "star-chain" sequence because the result of each step is used immediately in the next step—no intermediate stores are required. The sequence requires only two registers— the original multiplicand and the work register in which the product is formed. Consider the following examples, in which the notation *R1* indicates the multiplicand, <<= *n* means shift left *n* bit positions, and += *R1* means add the

	80286	68000	68020
ADD	2	8	0–3
SUB	2	8	0–3
SHIFT n	5+n	6+2n	1–4
MUL 16	21	70	21–28
MUL 32	**	**	41–44

Table 1: *Timing for several basic arithmetic instructions (clock cycles)*

multiplicand:

R1 * 10:
1 Rw = R1
2 Rw <<= 1
3 Rw += R1
4 Rw <<= 2
5 Rw += R1

R1 * 7:
1 Rw = R1
2 Rw <<= 1
3 Rw += R1
4 Rw <<= 1
5 Rw += R1
6 Rw <<= 1
7 Rw += R1

Note that the shifts and additions always come in pairs. Note, also, that there are as many shift-add pairs as there are one bits in the multiplier. This implies that the worst-case sequence will have as many shift-adds as the bit width of the multiplier.

You can generate shorter sequences by using shift-subtract as well as shift-add pairs. If the notation $2\ \hat{n}$ indicates 2 to the power n, you can write $((2\ \hat{n}) - 1)$ to denote a binary integer with n 1s in a row (for example, 8 − 1 = 7). Hence, the sequence shown above can be shortened to:

R1 * 7:
1 Rw = R1
2 Rw <<= 3
3 Rw −= R1

Here one shift-subtract replaces three shift-adds. The worst case is now alternating 1s and 0s in the multiplier, requiring at most one-half the number of sequence steps.

A further improvement can be made in the algorithm. Some numbers (such as 55 and 119) have a series of 1s, then a single 0, then another series of 1s (119 = 1110111 binary). The algorithm would generate a shift-subtract, then a shift-add by one place. Here is the sequence for 119:

R1 * 119:
1 Rw = R1
2 Rw <<= 3
3 Rw -= R1
4 Rw <<= 1
5 Rw += R1
6 Rw <<= 3
7 Rw -= R1

```
#include stdio.h
/*Program to generate a "star-chain" sequence to replace
    multiplication by a positive integer constant with a
    series of add, subtract, and shift-left instructions.
    Assumes two machine "registers". Instructions are
    formed on a temporary stack, then output. A stack
    element's magnitude is the shift amount, the sign
    indicates subsequent add (plus) or subtract (minus). */

long mult; /* 32-bit signed constant multiplier */
int flag,cnt,stkptr,stack[16], last_cnt,last_shift,ts;

int trim_trailing(one_zero) int one_zero;
{
    int c;
    for (c=0; ((mult & 1) = one_zero); c++, mult >>=1);
    return c;
}
main()
{
    stkptr = 0; /* init. stack pointer */
    printf("\nenter integer multiplier"); scanf("%d",&mult);
    if (mult > 0)
    {
       last_cnt = 0;
       last_shift = trim_trailing(0); /* cut trailing 0's */
       while (1)
       {/* decompose "mult", build stacked instructions */
            cnt = trim_trailing(1); /* count low-order 1's */
            if (cnt > 1)
            {/* more than 1 bit, use shift-subtract */
               flag = 0;
               if (last_cnt == 1)
               /* shift k, sub, shift 1, add -->
                    shift k+1, sub */
                 /* overwrite last entry */
                 stack[stkptr - 1] = -(cnt+1);
               else
                 stack[stkptr++] = -cnt;
            }

            /* will need another shift-add */
            else flag = 1;

            /* "mult" fully decomposed, time to output */
            if (mult == 0) break;

            /* count low-order zeros */
            last_cnt = trim_trailing(0) + flag;
            stack[stkptr++] = last_cnt; /* shift-add */
       }

/*now output code from stack */

    print("\nRw = R1"); /* load working register */
    while (stkptr > 0)
    {
       ts = stack[--stkptr]; /* get top stack element */
       if (ts < 0) printf("\nRw <<= %d \nRw -= R1",-ts);
       else             printf("=\nRw <<=%d \nRw +=R1",ts);
    }

    if (last_shift != 0) printf("\nRw <<= %d", last_shift);
    }
    else
    printf("\=\nRw = 0"); /* special case for mult = 0 */
}
```

***Code Example 1:** The star-chain algorithm in C*

Dr. Dobb's Journal of Software Tools . . .
the R_x *for programmers*

SUBSCRIBE AND SAVE

Subscribe to **Dr. Dobb's Journal**
and save over $5—a 15% savings
off the cover price!

___Please charge my: ___ Visa ___ MC ___ American Express
___ Payment enclosed ___ Bill me later

Card # ________________ Exp. date ________
Signature ________________
Name ________________
Address ________________
City ________________ State ______ Zip________

ONLY $29.97

Canada & Mexico add $10 for surface mail. All other countries add $27 for airmail. All foreign subscriptions must be pre-paid in U.S. dollars drawn on a U.S. bank. Please allow 6–8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc.

3440

SUBSCRIBE AND SAVE

Subscribe to **Dr. Dobb's Journal**
and save over $5—a 15% savings
off the cover price!

___Please charge my: ___ Visa ___ MC ___ American Express
___ Payment enclosed ___ Bill me later

Card # ________________ Exp. date ________
Signature ________________
Name ________________
Address ________________
City ________________ State ______ Zip________

ONLY $29.97

Canada & Mexico add $10 for surface mail. All other countries add $27 for airmail. All foreign subscriptions must be pre-paid in U.S. dollars drawn on a U.S. bank. Please allow 6–8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc.

3440

Comments & Suggestions

Dear Reader, **March 1987, #125**

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail.

—Ed.

Which articles or departments did you enjoy the most this month? Why?

____

Comments or suggestions: ________________

Name: ________________
Address: ________________

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. BOX 27809
SAN DIEGO, CA 92128

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. BOX 27809
SAN DIEGO, CA 92128

PLACE STAMP HERE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

501 GALVESTON DR.
REDWOOD CITY, CA 94063

Dr. Dobb's Journal of Software Tools . . . the R_x for programmers

INTEGER MULTIPLICATIONS
(continued from page 35)

Steps 2 through 5 can be combined by incrementing the shift count in step 2 and omitting steps 4 and 5, giving the following sequence for *R1 * 119*:

```
R1 * 119:
1  Rw = R1
2  Rw <<= 4
3  Rw -= R1
4  Rw <<= 3
5  Rw -= R1
```

This sequence (32-bit operands) would require about 46 clocks on a 68000, which is faster than a single 16-bit multiplication. It would require five words of code, as compared to the two or three words required for a subroutine call. It seems clear that star-chain sequences can provide a way to readily optimize multiplication by a constant.

An Actual Implementation

The C program shown in Code Example 1, page 35, implements the star-chain algorithm. It prompts the user for the multiplier (which must be positive) and prints out the star-chain sequence. It would be easy to convert the program to generate code steps for use in an optimizing compiler.

The program works in two steps: the first step builds the sequence on a last-in, first-out stack; the second step outputs the sequence from the stack. Note that, because the multiplier is 32 bits long, the stack need only hold 16 elements; there is no danger of overflow. Each stack element holds a shift-add or shift-subtract. The encoding uses the sign of the stack element to indicate shift-add (plus) or shift-subtract (minus). The magnitude of the stack element is the shift count. The function *trim_trailing* is used to count the number of low-order 0s or 1s in the multiplier. Note that, as written, *mult* must be a global variable because *trim_trailing* operates on it. The variable *flag* is used to signal the shift-subtract optimization.

The program shown works only for positive multipliers, which is always the case in array addressing. To make it handle negative multipliers, simply call it with the absolute value of the multiplier and then output a "negate" instruction.

The sequences that this program produces are not unique. For example, *R1 * 119* can be written:

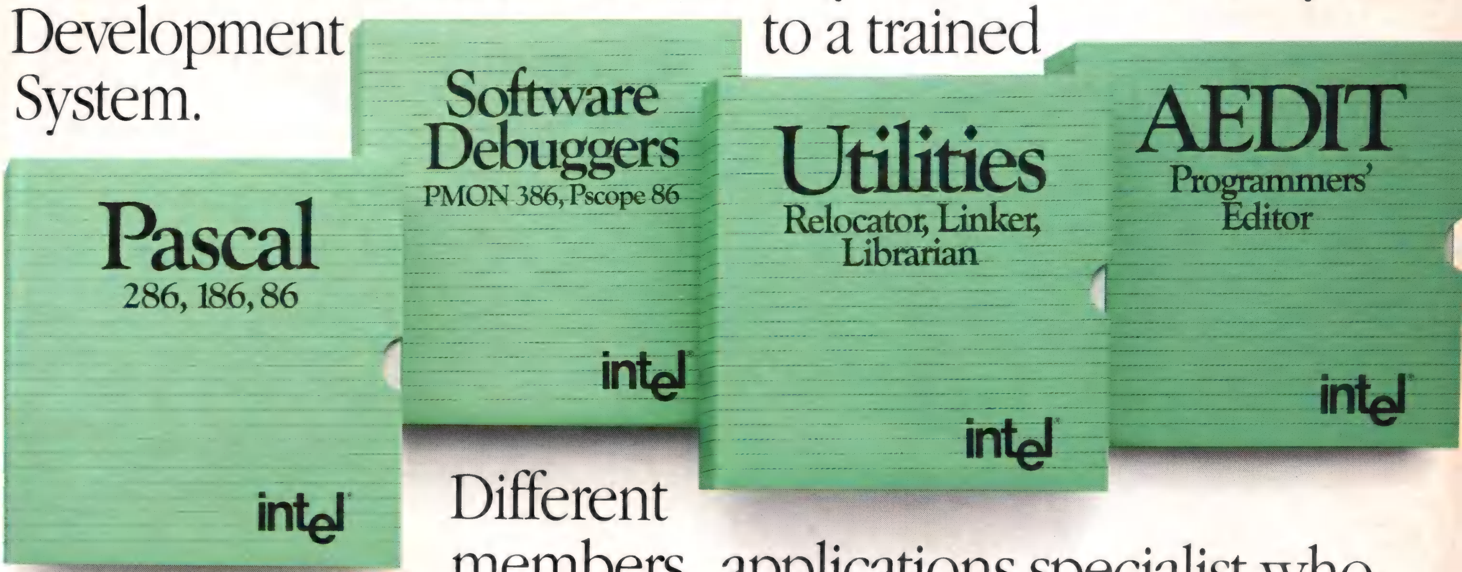
```
R1 * 119:
1  Rw = R1
2  Rw <<= 3
3  Rw -= R1
4  R1 = Rw
5  Rw <<= 4
6  Rw += R1
```

This sequence is derived by factoring 119 = 7 × 17. Steps 1 through 3 are a multiplication by 7, and steps 4 through 6 multiply by 17. The alternate sequence here is not shorter or faster than the one generated by the algorithm, but factoring can yield improvements in some cases. (Note that the multiplicand register *R1* is overwritten in step 4. The star-chain algorithm described in this article does not destroy the multiplicand.) The problem with the factoring approach is that it can take a great deal of time to find the factors (or it may require a large table of factorizations).

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

produce identical object code regardless of the host, you can write code at a PC running DOS, a VAX*/VMS terminal, or an Intel Development System.



Different members of the same design team can therefore choose the most effective combination of languages and systems to get the job done faster.

Intel post-sales support can also help you get the job done faster. We invented the microprocessor. We know microprocessors and languages for Intel architectures better than anyone else.

When you buy an Intel language, you have access to our customer hotline. So if you ever have a question you can talk directly to a trained applications specialist who understands our products. And can give you the right answers. Faster.

To order today, or get more information—including a free catalog of our development tools—call toll-free 1-800-87-INTEL.

The sooner you call, the faster you'll get the job done.

intel®

Circle **no. 179** on reader service card.

Listing One *(Text begins on page 16.)*

```
/* Listing one */
/* Subroutines for converting a pixel image
   to a quadtree and output the quadtree as
   locational codes

   Written by: Ronald G. White

   External routines:
      px2quad - only entry point
 */

#include <stdio.h>

/* Define structure for each node */
typedef struct qnode {
  struct qnode *child[4]; /* pointers to each child */
  struct qnode *next;     /* used during output */
  int color;
  int ntype;              /* see below for types */
  int locode;             /* location code */
} QNODE, *PNODE;

/* Node types: */
#define LEAF    1               /* no children */
#define BLEND   2               /* color of >2 kids */
#define WASH    3               /* color irrelevent */

extern getpix(); /* return pixel color at given posn */
extern putllc(); /* output location code and color */

static int toplevel; /* top level of tree (root node) */

px2quad(size)
int size;
/* entry point for these routines. Control routine to
   create a quadtree from the pixel image and output it
   as locational codes

   input:
      size - size of the image rounded up to nearest
      power of two
 */

{
        PNODE crtnode(), proot;

        /* Create the root node */
        proot = crtnode();

        /* Calculate the toplevel number */
        toplevel = 0;
        while (size > 1) {
                toplevel++;
                size >>= 1;     /* divide by two */
        }

        /* Build the quad tree */
        proot->locode = 1;
        addnode(proot, toplevel);

        /* Output it as location codes */
        outtree(proot);
}

static PNODE crtnode()
/* create a quadtree node and initialize it

   output:
      returns a pointer to the node
 */
{
      int i;
      PNODE newnode;

      /* Get space for it */
      newnode = (PNODE) malloc(sizeof(QNODE));
      if (newnode == NULL) {
        /* Something went wrong */
        fprintf(stderr,
        "crtnode: malloc failure; unable to continue0);
        exit(1);
```

```
	}

	/* Initialize it */
	for (i = 0; i < 4; i++) {
		newnode->child[i] = NULL;
	}
	newnode->color = 0;
	newnode->ntype = LEAF;
	return(newnode);
}

static addnode(pnode, level)
int level;
PNODE pnode;
/* add a new node to the quad tree
   If the node is not at the bottom level, four child
   nodes are created and added below the current node.
   Otherwise the node color is set to that of the
   corresponding pixel.

   input:
      pnode    - pointer to the current node
      level    - level number of the current node
 */

{
	int i;
	int newlevel;
	PNODE crtnode(), newchild;

	/* if this node is not at the bottom level,
	 * add four children below this node
	 */
	if (level > 0) {
	  newlevel = level - 1;
	  for (i = 0; i < 4; i++) {
	    newchild = crtnode();
	    pnode->child[i] = newchild;
	    newchild->locode = (pnode->locode << 2) + i;
	    addnode(newchild, newlevel);
	  }

	  /* Remove any unnecessary children */
	  condense(pnode);

	/* bottom level; get actual pixel color */
	} else {
	  pnode->color = getcolor(pnode->locode);
	}
}

static condense(pnode)
PNODE pnode;
/* examine children of the current node and
   remove any that are unnecessary

   input:
      pnode - pointer to current node
 */

{
   int colcnt[4];
   int colors[4];
   int i, j;
   int maxclr = 0;
   int nkids;
   int childclr;
   PNODE pchild;

   /* Initialization */
   for (i = 0; i < 4; i++) {
	colcnt[i] = 0;
   }

   /* Determine colors of children */
   for (i = 0; i < 4; i++) {
      pchild = pnode->child[i];
      if (pchild->ntype == WASH) {
	      /* this child has no color */
	      continue;
      }
   childclr = pchild->color;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 16.)*

```
   /* loop through colors found so far:
      do we have a match?
      note: we'll always "break" out of
      this loop because there can be at
      most four different colors.
    */

   for (j = 0; j < 4; j++) {
      if (colcnt[j] == 0) {
         /* new color */
         colors[j] = childclr;
         colcnt[j] = 1;
         break;

      } else if (childclr == colors[j]) {
         /* existing color */
         colcnt[j]++;
         if (colcnt[j] > colcnt[maxclr]) {
                maxclr = j;
         }
            break;
      }
   }
}

/* Set node color */
pnode->color = colors[maxclr];

/* Remove redundant children -- if more than
   one child node has the same color as the
   current node, then it contains redundant
   information.  If the redundant node is a
   leaf node, it can just be removed.  If it
   is not a leaf node, mark it as a WASH type
   and ignore it during output.
 */

nkids = 4;
if (colcnt[maxclr] > 1) {
     /* Loop through the four children */
     for (i = 0; i < 4; i++) {
          pchild = pnode->child[i];

          /* If child node is already a WASH,
             nothing else can be done to it
           */
          if (pchild->ntype == WASH) {
                 continue;
          }

          childclr = pchild->color;

          /* Check for color match */
          if (childclr == pnode->color) {

               /* If child is leaf, release */
               if (pchild->ntype == LEAF) {
                       relnode(pchild);
                       pnode->child[i] = NULL;
                       nkids--;

               /* otherwise, mark it as a WASH */
               } else {
                       pchild->ntype = WASH;
               }
          }
     }
}

/* Reset node type -- a LEAF node has no children */
if (nkids == 0) {
        pnode->ntype = LEAF;

/* A BLEND node has a color that represents some
   missing children, but still has some other
   children that are a different color.
 */
} else if (colcnt[maxclr] > 1) {
       pnode->ntype = BLEND;

/* A WASH node is necessary in the quadtree because
     it points to existent children nodes, but will not
     be output because its information (i.e. color) is
     available either in child nodes or parent nodes.
   */
  } else {
          pnode->ntype = WASH;
  }
}

relnode(pnode)
PNODE pnode;
/* release a node

   input:
      pnode - pointer to node to release
 */
{
  free((char *) pnode);
}

static getcolor(lcode)
int lcode;
/* get the color of the pixel corresponding to a
   bottom level node whose position is given by a
   locational code

   input:
      lcode - locational code of bottom level node

   output:
      returns pixel color
 */
{
  int dir;
  int col = 0;
  int level;
  int row = 0;
  int shift;

  /* Convert node locational code to pixel row & column
     by looping through direction codes in locational
     code for each level from top to bottom
   */
  for (level = toplevel; level > 0; level--) {

          /* shift last row & col values left one bit */
          col <<= 1;
          row <<= 1;

          /* calculate the position of the direction
             code for this level and extract it
           */
          shift = (level - 1) * 2;
          dir = (lcode >> shift) & 0x3;

          /* increment the col value if quadrant is in
             left half, i.e. NE or SE child
           */
          if (dir == 1 || dir == 3) {
                  col++;
          }

          /* increment the row value if quadrant is in
             bottom half, i.e. SW or SE child
           */
          if (dir == 2 || dir == 3) {
                  row++;
          }
  }

  /* return pixel color */
  return(getpix(col, row));
}

static outtree(proot)
PNODE proot;
/* output the relevant nodes in the quad tree
 *
 * proot - pointer to the root node
 */
{
```

(continued on page 44)

Clarify your source code

C, BASIC, Pascal, dBASE, Modula-2 programmers: be more productive with two new utilities from Aldebaran Laboratories

"Occasionally, a utility comes along that makes a programmer's life much easier. SOURCE PRINT is such a program. It contributes to the programmer's job by organizing code into a legible format and by helping to organize the documentation and debugging process."

— PC Magazine Sept. 16, 1986

Source Print and Tree Diagrammer both have easy-to-use menus with point-and-shoot file selection, and let you search for files containing a given string. For IBM PC and compatibles with 256K.

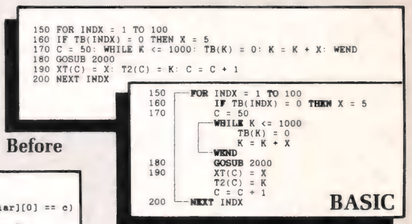
Join thousands of programmers who are working more efficiently using Source Print and Tree Diagrammer. Order these indispensable tools today. We ship immediately, and there's no risk with our 60-day money-back guarantee. **Order both and save. Only $115.00.**

800-257-5773 Dept. 53
In California:
800-257-5774 Dept. 53

MasterCard, VISA, American Express, COD. Add $5 for shipping.

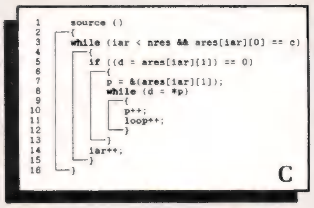
Source Print and Tree Diagrammer are trademarks of Aldebaran Labs. dBASE is a trademark of Ashton Tate.

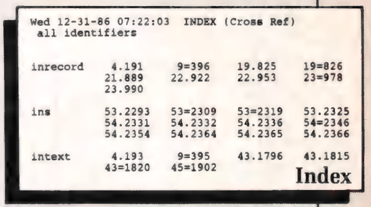
Source Print™



organizes your source code, simplifies debugging, and makes documentation a snap! It lists one or more source files with informative page headings and optional line numbers, while offering invaluable features:

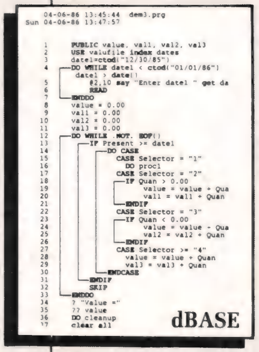
The Index (Cross-Reference list) saves you time by showing exactly where variables are used and where functions, procedures, and routines are called.





$75.00

Locations where new values may be assigned to variables are shown, making it easy to track down that mysterious value change.



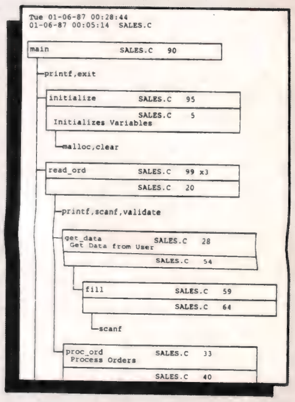
Structure Outlining solves the problem of hard-to-see nested control structures by automatically drawing lines around them.

Automatic Indentation of source code and listings reduces your editing time and ensures indentation accuracy.

Plus . . . Source Print generates a table of contents listing functions and procedures. Keywords can be printed in boldface on most printers. Multi-statement BASIC lines can be split for readability. Functions and procedures can be drawn by name from one or more source files to form a new file.

Tree Diagrammer™

shows your program's overall organization at a glance. Ordinary program listings merely display functions, procedures, and subroutines sequentially, but do not identify the relationships between these routines. Our revolutionary new Tree Diagrammer automatically creates an "organization chart" of your program showing the hierarchy of calls to functions, procedures, and subroutines. Recursive calls are indicated and designated comments in the source code will appear on the chart.



Tree Diagrammer helps you organize your program more logically. And you'll be amazed at how easy it is to debug when you see how your routines interact.

$55.00

Aldebaran Laboratories 3339 Vincent Rd. Pleasant Hill, CA 94523 415-930-8966

YES! Rush me ☐ Source Print @ $75. ______ ☐ Tree Diagrammer @ $55. ______
☐ Both $115. Ship/Handling $5. For CA add 6% tax ______ Total ______
Name ______
Company ______
Address ______
City ______ State ______ Zip ______
☐ Check enclosed ☐ VISA ☐ MasterCard ☐ American Express
Card # ______ Exp. Date ______
Signature ______ Phone # ______ 5C

Circle **no. 350** on reader service card.

COMPRESSING IMAGES

Listing One

(Listing continued, text begins on page 16.)

```
   PNODE outnode(), pcur, plast;

   /* Set up the linked list with root node */
   pcur = proot;
   plast = proot;
   proot->next = NULL;

   /* Output each node on the linked list in order
      until there are no more nodes on the list
    */
   while (pcur != NULL) {
           plast = outnode(pcur, plast);
           pcur = pcur->next;
   }
}

static PNODE outnode(pnode, plast)
PNODE pnode, plast;
/* output the locational code and color index for a
   node and put its children on the list

   input:
      pnode  - pointer to node to output
      plast  - pointer to last node on the linked list

   output:
      returns pointer to new last node on linked list
 */
{
  int i;
  PNODE pchild;

  /* If node is not a WASH, output it */
  if (pnode->ntype != WASH) {
          putlcc(pnode->locode, pnode->color);
  }

  /* Put the node's children on list */
  if (pnode->ntype != LEAF) {
          for (i = 0; i < 4; i++) {
                  pchild = pnode->child[i];
                  if (pchild != NULL) {
                          plast->next = pchild;
                          plast = pchild;
                  }
          }
  }

  /* Return new last pointer */
  return(plast);
}
```

End Listing One

Listing Two

```
/* Listing two */
/* Subroutines for displaying an image from a quadtree
   input as locational codes

   Written by: Ronald G. White

   External routines:
      qdisp - only entry point

 */

#include <stdio.h>

extern getnxn();            /* read in next node's data */
extern filrec();            /* fill rectangular region */

static int orgsize;         /* original image size */

qdisp(size)
int size;
/* main entry point for the display of a quadtree
 *
```

```
 * input:
 *     size - size of the original image
 */
{
     int lcode, color;
     int corner[2];
     int side;

     /* Make the image size global */
     orgsize = size;

     /* Read and display each node */
     while (getnxn(&lcode, &color) != EOF) {

       /* Convert loc code to corners, side of square */
       square(lcode, corner, &side);

       /* Fill in the square */
       filrec(corner[0], corner[1], side, side, color);
     }
}

square(lcode, corner, pside)
int lcode;
int corner[2];
int *pside;
/* convert quadtree locational code to corner and side
   of the square represented by the corresponding node.

   input:
      lcode - locational code for this node

   output:
      corner - upper left corner of quadrant
      pside  - the size of the quadrant in pixels
 */
{
     int dir;
     int shift;

     corner[0] = corner[1] = 0;
     *pside = orgsize;

     /* Find the begining of the code */
     for (shift = 30;
       ((lcode >> shift) & 0xff) == 0; shift -= 2);

     /* Convert node locational code to corner row &
        column by looping through direction codes in
        locational code for each level from top down.
      */
     for (shift -= 2; shift >= 0; shift -= 2) {

          /* The side of the square is reduced by a
             factor of two each level down.
           */
          *pside >>= 1;

          /* extract the direction code */
          dir = (lcode >> shift) & 0x3;

          /* increment the col value if quadrant is
             in left half, i.e. NE or SE child
           */
          if (dir == 1 || dir == 3) {
                  corner[0] += *pside;
          }

          /* increment the row value if quadrant is
             in bottom half, i.e. SW or SE child
           */
          if (dir == 2 || dir == 3) {
                  corner[1] += *pside;
          }
     }
}
```

End Listings

THE PROGRAMMER'S SHOP

helps save time, money and cut frustrations. Compare, evaluate, and find products.

RECENT DISCOVERY

Sapiens V8 - virtual memory management for C programmers on PCs provides 8M workspace. 64-bit emulation, virtual stack library and heap. Link to MS, Lattice, Aztec. PC $ 300

AI-Expert System Dev't

Arity System-incorporate with C programs, rule & inheritance MS $ 259
Auto-Intelligence - by IntelligenceWare PC $ 749
Experteach - Powerful, no limit on memory size. Samples PC $ 349
Intelligence/Compiler - rules, frames PC $ 749

AI-Lisp

Microsoft MuLisp 85 MS $ 179
PC Scheme LISP - by TI PC $ 85
TLC LISP - classes, **compiler.** MS $ 225
TransLISP - learn fast MS Call
TransLISP PLUS
Optional Unlimited Runtime $ 150
PLUS for MSDOS $ 179
Others: IQ LISP ($155), UNX LISP ($59), IQC LISP ($269), WALTZLISP ($139)

AI-Prolog

APT - **A**ctive **P**rolog **T**utor - build applications interactively PC $ 65
ARITY Standard - full, 4 Meg
Interpreter - debug, C, ASM PC $ 319
COMPILER/Interpreter-EXE PC $ 699
With Exp Sys, Screen - KIT PC $1129
LPA MacProlog Complete - incremental compiler and an interpreter MAC $ 295
LPA MicroProlog - intro MS $ 85
Prolog-86 - Learn Fast, Standard, tutorials, samples MS $ 89
Prolog-86 Plus - Develop MS $ 229
TURBO PROLOG by Borland PC $ 69

AI-Other

METHODS - SMALLTALK has objects, windows, PC $ 69
Q'NIAL - Combines APL with LISP. Source or binary. PC $ 349
Smalltalk-80 - Xerox improved PC $ 995
Smalltalk/V-graphics PC $ 89

Atari ST & Amiga

We carry full lines of Manx, Lattice, & Metacomco.
Amiga - LINT by Gimpel Amiga $ 79
Cambridge LISP Amiga $ 200
Lattice C ST, Amiga $ 139
Lattice Text Utilities Amiga $ 75
Megamax - tight, full ST $ 200

FEATURES

C Scape - capture Dan Bricklin's, 1-2-3, Turbo screens & more, convert to C. Plus full screen generation package - tiled, pop-up windows with scrolling, validation. Source PC $ 179

PolyBoost - Run 2 to 10 times faster with software accelerator. Speeds disk access, screen display, keyboard input. PC $ 69

National Accounts

MIS, Engineering, and Research departments get special FREE consulting, product comparisons, reports, newsletters. Compare approaches to COBOL, C, AI. PURCHASING AGENTS - get help and special service finding products, negotiating license agreements, with billing and more. Call 800-446-1185.

Our Services:

- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature FREE
- BBS - 7 PM to 7 AM 617-826-4086
- Dealers Inquire
- Newsletter
- Rush Order
- Over 700 products
- *National Accounts Center*

Basic

Basic Development System - for BASICA; Adds Renum, more. PC $ 105
Basic Development Tools by Sterling Castle PC $ 89
Basic Windows by Syscom PC $ 95
BetterBASIC - all RAM, modules Structure. Full BASICA PC $ 129
8087 Math Support PC $ 75
Run-time Module PC $ 169
Better Tools - for Better Basic PC $ 95
CADSAM FILE SYSTEM-full MS $ 69
GoodBas - maintain code PC $ 95
LPI Basic - MS compatible UNIX $1100
Prof. Basic - Interactive, debug PC $ 75
8087 Math Support PC $ 45
QuickBASIC V2.0-New interface PC $ **69**
TRUE Basic - ANSI PC $ 119
Run-time Module PC $ 129

Cobol

Macintosh COBOL - full MAC $ 459
MBP - Lev. II, native MS $ 819
Microfocus Professional Cobol PC $2295
VS Workbench PC $3379
Microsoft COBOL MS $ 439
Microsoft Cobol Tools - xref, debugger w/source support. PC $ 209
Realia - very fast MS $ 819
Ryan McFarland COBOL MS $ 649
COBOL-8X MS $ 895
Screenplay - by Flexus. Interactive screen mgmt. PC $ 175

Editors for Programming

BRIEF Programmer's Editor PC Call
EMACS by UniPress - powerful, multifile, windows Source:$929 $ 299
Epsilon - like EMACS, full C-like language for macros. PC $ 155
KEDIT - like XEDIT PC $ 105
Lattice Screen Editor - multiwindow, multitasking Amiga $ 89 MS $ 109
PC/EDT - macros PC $ 250
PC/VI - by Custom Software MS $ 109
Personal REXX PC $ 109
PMATE - power, multitask PC $ 119
SPF/PC - fast, virtual memory PC $ 139
XTC - multitasking PC $ 79

C Libraries-Communications

Asynch by Blaise PC $ 135
Essential Comm Library PC $ 135
With Debugger PC $ 199
Greenleaf Comm Lib. PC $ 139
Multi-Comm - add multitasking, use w/Multi-C PC $ 149
Software Horizons pack 3 PC $ 119

RECENT DISCOVERY

r-tree - report generation for ctree. Multiple file handling, fixed or variable length. Many built-in functions like Boolean, computational functions, string, date handling, numeric to string conversion. Layout control. Source in C. PC $249

C Language-Compilers

AZTEC C86 - Commercial PC $499
C86 by CI - 8087, reliable MS $299
Datalight C - fast compile, good code, 4 models, Lattice compatible, Lib source. Dev'rs Kit PC $ 77
HOT C - new, intriguing PC $ 85
Lattice C - from Lattice MS $289
Mark Williams - w/debugger MS $369
Microsoft C 4.0- Codeview MS $**279**
Wizard C MS $359

C Language-Interpreters

C-terp by Gimpel - full K & R MS $229
C Trainer - by Catalytix PC $ 89
INSTANT C - Source debug, Edit to Run-3 seconds, .OBJs MS $379
Interactive C by IMPACC Assoc. PC $225
Introducing C-self paced tutorial PC $105
Run/C Professional MS $179
Run/C Lite MS $ 97

C Libraries-General

Blackstar C Function Library PC $ 79
C Essentials - 200 functions PC $ 83
C Food by Lattice-ask for source MS $ 99
C Scientific Subroutines - Peerless MS $135
C Tools Plus (1 & 2) - Blaise PC $135
C Utilities by Essential - Comprehensive screen graphics, strings, source. PC $137
C Worthy Library - Complete, machine independent MS $269
Entelekon C Function Library PC $119
Entelekon Superfonts for C PC $ 45
Greenleaf Functions-portable, ASM $139
PforCe by Phoenix - objects PC $229

C Libraries-Files

FILES: C Index by Trio - full B + Tree, vary length field, multi compiler
/File is object only MS $ 89
/Plus is full source MS $319
CBTREE - Source, no royalties MS $ 99
CTree by Faircom - no royalties MS $319
dbQUERY - ad Loc, SQL - based MS $159
dbVISTA - full indexing, plus optional record types, pointers, Network.
Object only - MS C, LAT, C86 $155
Source - Single user MS $425
Source - **Multiuser** MS $845
dBASE Tools for C PC $ 65
dbc Isam by Lattice MS $179
dBx - translator MS $319
w/source to library MS $499

FEATURE

Uniware Cross Development Tools include 68000 C compiler. Development Package with compiler, assembler, link editor, and utilities, 17 cross assemblers for Intel, TI, Motorola, Zilog, etc. - relocatable, macros. MS Call

We support MSDOS (not just compatibles), PCDOS, Xenix-86, CPM-80, Macintosh, Atari ST, and Amiga.

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees and every product for Microcomputer Programming.

AI TOOLS

Expert System Shells and professional language implementations, smart decision support packages make practical AI power in your applications, planning. Call one of our AI specialists today.

Order before 3/31/87 and mention this ad for these special prices.

	List	Normal	SPECIAL
Insight 2 + Expert Shell	$ 485	$ 389	**$ 309**
LPA MicroProlog Professional	$ 395	$ 339	**$ 289**
Arity Compiler with Expert Shell Interpreter	$ 795	$ 699	**$ 589**
Exsys	$ 395	$ 309	**$ 259**
Texas Instruments PC Easy	$ 495	$ 439	**$ 349**
Texas Instruments Personal Consultant Plus	$2950	$2599	**$1899**

C Support-Systems

Basic-C Library by C Source	PC	$139
C Sharp - **realtime,** tasks.	PC	$600
C ToolSet - DIFF, xref, source	MS	$ 95
The HAMMER by OES Systems	PC	$149
Lattice Text Utilities	MS	$ 89
Multi-C - multitasking	PC	$149
PC LINT-Checker. Amiga $89	MS	$107
SECURITY LIB-add encrypt to MS C, C86 programs. Source $229	PC	$115
Quickshell - script compiler	PC	$349

C-Screens, Windows, Graphics

C Power Windows by Entelekon	PC	$109
dBASE Graphics for C	PC	$ 69
Curses by Lattice	PC	$ 89
ESSENTIAL GRAPHICS - fast	PC	$199
GraphiC - mono version	PC	$217
GraphiC - new color version	PC	$295
Greenleaf Data Window	PC	$189
w/source	PC	$339
Multi-Windows - use w/ Multi-C	PC	$295
Screen Ace Form Master	PC	$195
Vitamin C - screen I/O	PC	$199
Windows for C - fast	PC	$159
Windows for Data - validation	PC	$239
ZView - screen generator	MS	$189

Debuggers

386 Debug	PC	$149
Advanced Trace-86 by Morgan Modify ASM code on fly.	PC	$125
CODESMITH - visual, modify and rewrite Assembler	PC	$107
C SPRITE - data structures	PC	$129
DSD87 - by Soft Advances	PC	$ 89
Periscope I - own 16K	PC	$239
Periscope II - symbolic, "Reset Box," 2 Screen	PC	$109
Periscope II-X - software only	PC	$ 85
Pfix-86 Plus Symbolic Debugger by Phoenix - windows	PC	$229
Showcase by Test Software	PC	$135
Software Source by Atron - Lattice, MS C, Pascal, Windows single step, 2 screen, log file.	MS	$115
w/Breakswitch		$199

FEATURE

Multi-Comm - Multitasking/user communications. Use with Cytek's Multi-C; interrupt-driven, ROMable; XMODEM, ASCII or binary transfer. MS $149

Fortran & Supporting

50:More FORTRAN - math, source	PC	$ 99
ACS Time Series	MS	$419
Forlib + by Alpha	MS	$ 59
MACFortran by Microsoft	MAC	$229
MS Fortran link to C	MS	$**209**
No Limit - Fortran Scientific	PC	$115
RM/Fortran	MS	$389
Scientific Subroutines - Matrix	MS	$139
Statistician by Alpha	MS	$249
Strings and Things - register, shell	PC	$ 55

Multilanguage Support

BTRIEVE ISAM	MS	$199
BTRIEVE/N-multiuser	MS	$469
CODESIFTER - Profiler.	MS	$ 99
HALO Graphics - 115 + devices. Animation, engineering, business. Any MS language, Lattice, C86	PC	$209
Informix - by RDS	PC	$639
Informix 4GL - application builder	PC	$799
Opt Tech Sort - sort, merge	MS	$119
PANEL	MS	$219
PolyLibrarian by Polytron	MS	$ 79
PVCS Version Control	MS	$329
QMake by Quilt Co.	MS	$ 84
Rtrieve - Xtrieve option	MS	$119
Screen Sculptor - slick, thorough, fast, BASIC, PASCAL.	PC	$ 95
Xtrieve - organize database	MS	$199
ZAP Communications - VT 100	PC	$ 89

Pascal and Supporting

ALICE - learn Pascal, Turbo compatible, interpreter	PC	$ 68
Exec - Chain Programs	MS	$ 79
MetaWINDOW - graphics toolkit	PC	$115
MetaWINDOWS PLUS - enhanced	PC	$185
Microsoft PASCAL - faster	MS	$189
MICROTEC PASCAL - 5 memory models, "Iterators", 65 bit 8087 strings	MS	$665
Pascal Extender - access user interface, toolbox	MAC	$ 65
Pascal Pac with Tidy - formatter, utilities	PC	$ 69
Pascal Tools - strings, screen	PC	$109
Pascal Tools 2 - by Blaise	MS	$ 85
Pascal 2 - tight, fast	MS	$329
TurboHALO - 150 routines, IBM EGA, Hercules, more	PC	$ 85

RECENT DISCOVERY

The Documentor - for dBASE program flow chart, tree diagrams, .DBF documentation, variable/field concordance, hierarchy charts. Macros, searches, configure options. MS $ 295

Other Languages

APL*PLUS/PC	PC	$ 469
CLIPPER-dBASE Compiler	MS	$ 429
dBXL by Word Tech	PC	$ 129
Lattice RPF II Compiler	PC	$ 719
MasterForth - Forth '83	MAC or PC	$ 109
Microsoft MASM - faster	MS	$ 98
Modula-2/86 Compiler by Logitech w/ 8087 ($ 99), 512K ($145).	PC	$ 62
Pasm - by Phoenix	MS	$ 109
PC Forth + - by Laboratory Microsystems	PC	$ 205
SNOBOL4 + - great for strings	MS	$ 85
Turbo Edit/ASM - by Speedware	PC	$ 85

Xenix-86 & Supporting

Basic - by Microsoft	$ 239
Cobol - by Microsoft	$ 639
Cobol Tools - by Microsoft	$ 319
Fortran or Pascal - by Microsoft	$ 439
MicroFocus Lev. II Compact COBOL	$ 795
Panel	$ 539
RM/Cobol	$ 949
RM/Fortran	$ 549
Xenix Complete System	$1049

Other Products

386 Assembler/Linker	PC	$ 459
ASMLIB - 170 + routines	PC	$ 129
BSW Make - like UNIX make	MS	$ 85
Compact Source Print	PC	$ 59
Dan Bricklin's Demo Program	PC	$ 59
dBrief - Customize BRIEF for dBASE development. with BRIEF $275.	PC	$ 95
H Test/H Format - XT Fix	PC	$ 89
Help/Control - on line help	PC	$ 109
Interactive Easyflow-HavenTree	PC	$ 129
Link & Locate - tools to work with Intel and Tektronix projects.	MS	$ 329
LMK - like UNIX make	MS	$ 139
Microsoft Windows	PC	$ 69
Software Development Kit	PC	$ 329
MKS Toolkit - Unix, vi, awk	PC	$ 119
PDisk - cache, tree	PC	$ 89
PMaker - by Phoenix	PC	$ 79
Polymake by Polytron	MS	$ 79
PS MAKE by UniPress	MS	$ 79
SECRET DISK by Lattice	PC	$ 89
SoftEst - Manage projects.	MS	$ 350
Source Print - by Aldebaran	PC	$ 89
Synergy-Create user interfaces	MS	$ 375
Texsys - control source	MS	$ 89
Tom Rettig's Library - dBASE	PC	$ 89
Tree Diagrammer	PC	$ 59
Visible Computer: 8088	PC	$ 65

Circle **no. 133** on reader service card.

Note: All prices subject to change without notice.
Mention this ad. Some prices are specials. Ask about COD and POs. Formats: 3" laptop now available, plus 200 others. UPS surface shipping add $3/item.

Call for a catalog, literature, advice and service you can trust

HOURS
8:30 AM - 8:00 PM EST.

800-421-8006

THE PROGRAMMER'S SHOP™
5-D Pond Park Road, Hingham, MA 02043
Mass: 800-442-8070 or 617-826-7531

"YOUR SERVICE IS FABULOUS!!! . . . Your information packets are so enticing; I really want 10 of them. . . Keep up the good work."
— Dan Bredemeyer
Bredemeyer

Circle **no. 133** on reader service card.

Listing Ten *(Text begins on page 96.)*

```
---------------------------------------------------------------------------
#include <stdio.h>
#include <stdarg.h>

ferr( fmt )
char    *fmt;
{
        /* ferr() is used for fatal error processing. It
         * is used just like printf(). However, it exits
         * the program with a status of 1 immediately after
         * printing the message. I'm using ANSI, not UNIX
         * variable argument conventions here.
         */

        va_list   args;
        va_start( args, fmt );
        vfprintf( stderr, fmt, args );
        exit( 1 );
}
```

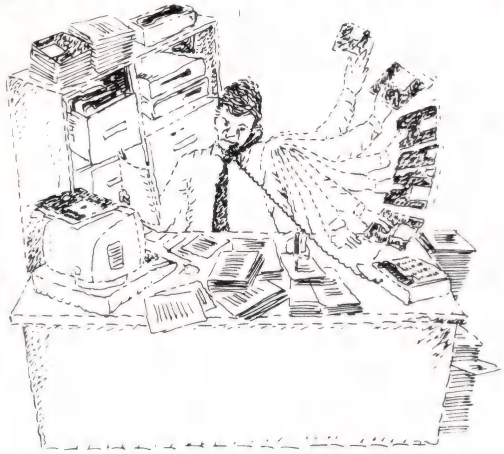
End Listing Ten

Listing Eleven

```
#include <ascii.h>

#define min(a,b)      ((a) < (b) ? (a) : (b))
#define max(a,b)      ((a) > (b) ? (a) : (b))

typedef unsigned char UCHAR;


#ifdef DEBUG                 /* For debugging, expands to it's argument when */
#       define D(x)   x      /* DEBUG is true, otherwise expands to an       */
#else                        /* empty string.                                */
#       define D(x)
#endif

#define MAXLTRAP 100         /* Largest line number on which we can set a line
                              *      trap.
                              */

#define MAXSTR   257         /* Maximum string width (this will limit both the
                              *      input and output widths
                              */

#define MAXPAGE  511         /* Max page number which can be given with the
                              * "-o" command line switch
                              */
#define MAXARGS  10          /* Max # of arguments in a macro                */
#define MAXNEST  10          /* Max level of macro nesting                   */
#define MAXMBUF  256         /* Largest macro stored in memory. Larger       */
                             /* macros are written to files.                 */

/*---------------------------------------------------------------------------
 *      Special characters:
 *
 *      These symbols are used internally to pass information
 *      from the character-oriented input functions to the
 *      (in nrinp.c) to the multiple-byte character processing
 *      functions (in nrtext.c and nrout.c). They are all
 *      two-character sequences. Of these, VMOVE, HMOVE, and
 *      CH_FONT are also used in the 16-bit wide CTYPE characters
 *      discussed below.
 */

#define VMOVE          ( 0xf8 )  /* Vertical motion                           */
#define HMOVE          ( 0xf9 )  /* Horizontal motion                         */
#define CH_FONT        ( 0xfa )  /* Change font                               */
#define CH_ATTRIB      ( 0xfb )  /* Change attribute in current font          */
#define SOFT_HYPHEN    ( 0xfc )  /* A soft hyphen goes here.                  */
#define ZWIDTH         ( 0xfd )  /* Next character is  zero width             */
#define UP_SPACE       ( 0xfe )  /* Unpaddable space                          */
#define LITCHAR        ( 0xff )  /* next character goes to printer is         */
                                 /* literal (it goes to the printer           */
                                 /* unchanged.                                */

/*---------------------------------------------------------------------------
 * Default fonts and attributes:
 *
 * BOLD, OVER, and ITALICS are attributes which may apply to any font. PREVIOUS
 * turns off these attributes but doesn't spring a change font macro. ROMAN
 * replaces the current font with the roman font and also clears all
 * attributes.
 */

#define BOLD        'B'   /*      Bold face       */
#define OVER        'O'   /*      Overstrike      */
#define ITALICS     'I'   /*      Italics         */
#define PREVIOUS    'P'   /*      Previous        */
#define ROMAN       'R'   /*      Roman           */

/*-------------------------------------------------------------------------*/

#define BOTH        'b'   /* Legal adjustment modes                        */
#define ALT_BOTH    'n'
#define LEFT        'l'
#define RIGHT       'r'
#define CENTER      'c'
```


(continued on page 53)

SOFTSTRIP®
NOW OFFERS YOU SOMETHING NEVER BEFORE AVAILABLE...



CONVENTIONAL DATA HANDLING

THE SOFTSTRIP SYSTEM

A CHOICE.

Until now you were stuck with disks.

No more. Install our unique STRIPPER™ software on your personal computer today and discover the many benefits of the fastest, easiest, least expensive way to handle information.

STRIPPER lets you print – ON PAPER – your own machine readable Softstrip data strips using your dot matrix printer. The Softstrip System Reader reads that information into a computer rapidly. With STRIPPER and the reader, your PC and printer become part of the most versatile information handling system available.

With this system you can do anything you wish with any data you have in your PC – ON PAPER.

DATA ENTRY: Why use keystrokes when you can eliminate them with data strips? Whatever the document - invoices, packing slips, memos, letters, sales reports, the list is endless — simply print a data strip right on the same printed page. Now you have a document that is both human readable and machine readable. A typical document can be entered in only 15 seconds using data strips. And, it ends keystroke errors forever.

DATA DISTRIBUTION: Why copy disks? It's time consuming and expensive. Softstrip data strips will end all that. Simply photocopy as many data strips as you like and send them by mail. Data strips ignore folding, coffee stains, ink marks and, by the way, magnetic fields. And if you're using telecommunications, you can stop making the phone company rich.

DATA STORAGE AND RETRIEVAL: Why have a file of disks and a file of paper? Eliminate one with Softstrip data strips. File the data strip with the document. Better still, print the strip right on the document. Then put it in a file or binder.

Retrieval is simple. To find existing data, pull the document and its related data strip from the file. They've been stored together. Then use the reader to enter the data. No more hassle trying to match documents with the right disk — if you can find it.

DATA TRANSFER: Why bother with cables, modems and phone lines to move files between computers? A Softstrip data strip generated by an IBM PC can be read into another PC, or compatible, an Apple or even a Macintosh. If you work at home on a Macintosh, make a data strip on your printer, take it into the office and read it into your IBM PC. Simple. And we've created the utilities to let you do that easily. (See Application Notes on opposite page.)

Fascinating, isn't it? Anything you can do with disks can be done with the Softstrip data strip system — faster, easier and at lower cost — ON PAPER.

All you need is STRIPPER software at **$19.95** and the Softstrip System Reader at **$199.95.**

Circle no. **334** on reader service card.

Listing Eleven *(Listing continued, text begins on page 96.)*

```c
/*-------------------------------------------------------------------------
 * Number registers:
 *
 *      The number registers are maintained as an array pointers to NREG
 *      type objects. The array is kept sorted by register name.
 *      nname is a pointer to the name, it usually points at nbuf.
 */

typedef struct _nr
{
        char            nname[3];
        int             nfmt    ;
        int             nval    ;
        int             incr_amt;
} NREG ;

                        /*  Read/write pre-defined number registers:   */
                        /*                                              */
extern  NREG    *Nrpg;  /* %   Page number                              */
extern  NREG    *Nrdy;  /* dy  Day                                      */
extern  NREG    *Nrdw;  /* dw  day of the week (0 = sun, 6= sat)        */
extern  NREG    *Nrh;   /* h   hour                                     */
extern  NREG    *Nrln;  /* ln  Current .nm line number                  */
extern  NREG    *Nrnl;  /* nl  Current output line number               */
extern  NREG    *Nrm;   /* m   minute                                   */
extern  NREG    *Nrmo;  /* mo  Month                                    */
extern  NREG    *Nrs;   /* s   second                                   */
extern  NREG    *Nryr;  /* yr  Year                                     */

                          /*  Read only pre-defined number registers:   */
                          /*                                            */
extern  NREG    *Nrhp;    /* hp  Current horizontal place on input line */
extern  NREG    *Nrdn;    /* dh  Height of most recent diversion        */
extern  NREG    *Nrdl;    /* dl  Width of last completed diversion      */
extern  NREG    *Nrargs;  /* .$  Number of args at current macro level  */
extern  NREG    *Nrlines; /* .c  Number of lines read from current input */
extern  NREG    *Nrvplace; /* .d Current vert. place in current diversion */
extern  NREG    *Nrfont;  /* .f  Index in Fonts[] of current font       */
extern  NREG    *Nrindent; /* .i Current indent column                  */
extern  NREG    *Nrllen;  /* .l  Current line length                    */
extern  NREG    *Nrtlen;  /* .n  Length of text portion of previous line */
extern  NREG    *Nroffset; /* .o Current page offset                    */
extern  NREG    *Nrplen;  /* .p  Current page length                    */
extern  NREG    *Nrtotrap; /* .t Distance to next trap                  */
extern  NREG    *Nrfill;  /* .u  1 if in fill mode, 0 otherwise         */
extern  NREG    *Nrv;     /* .v   current vertical base-line spacing    */
/*
 *      Number register format types. The specified character found in the
 *      left-most postion of the .af command's c argument signifies the
 *      type. In addition, the number of characters in the arabic padded mode
 *      determines the fieldwidth of the number.
 *      The format for arabic numbers is an ascii digit. If this digit is
 *      '0' or '1' then the number is printed unpadded. If the digit is a '4'
 *      it is printed in a 4 space field, right justified in the field and
 *      padded with zeros. The special format READONLY is used by the read
 *      only pre-defined number registers. They are always arabic format.
 */

#define ARABIC    '1'   /*   0,   1,   2,   ...                   */
#define PADDED    '0'   /* 000, 001, 002,   ...                   */
#define LC_ROMAN  'i'   /*   0,   i,  ii, iii, iv, v, ...         */
#define UC_ROMAN  'I'   /*   0,   I,  II, III, IV, V, ...         */
#define LC_ALPHA  'a'   /*   0,   a,   b, ... , z, aa, ab ...     */
#define UC_ALPHA  'A'   /*   0,   A,   B, ... , Z, AA, AB ...     */
#define LC_ENG    'e'   /* zero, one, two, three ...              */
#define UC_ENG    'E'   /* Zero, One, Two, Three ...              */
#define READONLY  'r'   /* Pre-defined, arabic format only        */
/*-------------------------------------------------------------------------
 *
 *      Default values of the pre-defined number registers
 */

#define DEF_PAGE        1       /*      Page number                     */
#define DEF_WIDTH       0       /*      Width of most recent diversion  */
#define DEF_HEIGHT      0       /*      Height of most recent diversion */
#define DEF_DAY         1       /*      Default day                     */
#define DEF_HORIZ       1       /*      Current place on input line     */
#define DEF_LINE        1       /*      Output Line number              */
#define DEF_MONTH       1       /*      Default month                   */
#define DEF_YEAR        1985    /*      Default year                    */
#define DEF_NARGS       0       /*      # Args in current macro         */
#define DEF_INLINES     0       /*      # Lines read from current input */
#define DEF_VERT        1       /*      Vertical place in current diver.*/
#define DEF_FONT        0       /*      Index in Fonts[] of current font*/
#define DEF_INDENT      0       /*      Current indent column           */
#define DEF_LINLEN      80      /*      Default line length             */
#define DEF_TEXTLEN     0       /*      Len of text part of previous line */
#define DEF_OFFSET      0       /*      Page offset                     */
#define DEF_PGLEN       66      /*      Page length                     */
#define DEF_TOTRAP      66      /*      Distance to next trap           */
#define DEF_FILL        0       /*      1 if in fill mode               */
#define DEF_LS          1       /*      Default line spacing            */
/*-----------------------------------------------------------------------*/

#define NUMTABS (MAXSTR + 1)        /* Largest column in which a tab can be set */
typedef int TSTOP[ NUMTABS ];
extern  TSTOP Tabstop;              /* The tabstop array (see nrglbls.c)       */
/*-------------------------------------------------------------------------
 *      Table used by command() to parse command lines:
 */

typedef char    *CHARPTR;
```

(continued on next page)

Listing Eleven *(Listing continued, text begins on page 96.)*

```
typedef struct
{
        char     *cmd;              /* Command name                        */
        int      (*action)();       /* Subroutine to call when cmd found */
        unsigned type  : 3 ;        /* Command type                        */
        unsigned inhib : 1 ;        /* 1 --> Inhibit works                 */
        char     *def;              /* Default value of numeric argument */
}
CTAB;

/*------------------------------------------------------------------
 *   Table of user defined fonts (Fonts) is made up of FONT type.
 *   NUMFONTS is the maximum number of user defined fonts.
 *   The "widths" array holds the character widths. Maximum
 *   number of characters is MAX_CHARS_IN_FONT. Font numbers
 *   must be single characters (In the range 0-9) so NUMFONTS
 *   must be <= 10. The "resolution" field is the middle argument
 *   from the .hd command that was in effect when the .df was
 *   executed. If a space is 6 units wide and "resolution" is set to
 *   2, then sending Right_str to the printer three times will
 *   move the carriage one space to the right. If "resolution" is
 *   1, then the string will have to be sent 6 times to move the
 *   same amount.
 */

typedef struct
{
        UCHAR    name;              /* Font name                          */
        UCHAR    smac[3];           /* Macro to enter new font            */
        UCHAR    emac[3];           /* Macro to exit  font                */
        int      resolution;        /* Min horizontal resolution from .hd */
        UCHAR    *left;             /* String to go left from .hd         */
        UCHAR    *right;            /* String to go right from .hd        */
        UCHAR    *widths;           /* Array of character widths.         */
}
FONT;

#define NUMFONTS            10
#define MAX_CHARS_IN_FONT   256

/*--------------------------------------------------------------------------
 *      Msc #defines and typedefs
 */


#define ISCMD(c) ( (c)==Cmd_chr || (c)==Nobreak )


/*--------------------------------------------------------------------------
 *      #Defines to get at the value fields of the pre-defined
 *      number registers (Those marked with an E are saved with a .ev
 *      command):
 */

                                                                             % */
#define PAGE     (Nrpg->nval)      /* Page number                              dl */
#define WIDTH    (Nrdl->nval)      /* Width of last completed diversion        dn */
#define HEIGHT   (Nrdn->nval)      /* Height of last completed diversion       dy */
#define DAY      (Nrdy->nval)      /* Day                                      hp */
#define HORIZ    (Nrhp->nval)      /* Current horiz-> place on input line      ln */
#define LINE     (Nrln->nval)      /* Current .nm line number                  nl */
#define OLINE    (Nrnl->nval)      /* Current output line number               mo */
#define MONTH    (Nrmo->nval)      /* Month                                    yr */
#define YEAR     (Nryr->nval)      /* Year                                     .$ */
#define NARGS    (Nrargs->nval)    /* # of args at current macro level         .c */
#define INLINES  (Nrlines->nval)   /* # of lines read from current input       .d */
#define VERT     (Nrvplace->nval)  /* Vert. place in current diversion         .f */
#define CURFONT  (Nrfont->nval)    /* Currently active font (Font[i])      (E) .i */
#define INDENT   (Nrindent->nval)  /* Current indent column                (E) .l */
#define LINLEN   (Nrllen->nval)    /* Current line length                  (E) .n */
#define TEXTLEN  (Nrtlen->nval)    /* Length of text part of prev out line     .o */
#define OFFSET   (Nroffset->nval)  /* Current page offset                  (E) .p */
#define PGLEN    (Nrplen->nval)    /* Current page length                      .t */
#define TOTRAP   (Nrtotrap->nval)  /* Distance to next trap                    .u */
#define FILL     (Nrfill->nval)    /* 1 if in fill mode, 0 otherwise       (E) .v */
#define LSPACE   (Nrv->nval)       /* Current line spacing (set w/.ls)     (E)    */
                                                                             wd */
#define WEEKDAY  (Nrdw->nval)      /* day of the week                           h */
#define HOUR     (Nrh->nval)       /* hour                                      w */
#define MIN      (Nrm->nval)       /* minute                                    s */
#define SEC      (Nrs->nval)       /* second
/*--------------------------------------------------------------------------
 *      Global vars used by more than one module. (Those marked with an E are
 *      saved with a .ev command.) Most of these are declared in nrglbls.h
 *      but some are command-line switches and are found in nr.c.
 */
                                                                          E */
extern  int  Adjmode     ; /* Current adjustment mode                     E */
extern  int  Adjusting   ; /* One if adjusting lines                      E */
extern  int  Cmd_chr     ; /* Command character                           E */
extern  int  Cont_ul     ; /* Num of input lines to continuously underline E */
extern  int  Divtrap     ; /* Location of diversion trap, -1 if none        */
extern  char Dtrap_name[]; /* Macro to invoke when diversion trap reached   */
extern  int  Divwidth    ; /* Width of most of last completed diversion     */
extern  int  Esc         ; /* Current escape character                    E */
extern  int  Hyphenate   ; /* Hyphenation is enabled during filling only    */
extern  int  Hyphen_chr  ; /* Soft hyphen is \<Hyphen_chr>. default = \%    */
extern  char *Endm       ; /* Name of macro invoked at end of input         */
extern FONT  Fonts[]     ; /* Table of user defined fonts                   */
extern  int  H_units     ; /* Number of horizontal units / inch             */
extern  int  H_space     ; /* Number of horizontal units in a space         */
extern FILE  *Ifile      ; /* Current input file                            */
extern  char *Ifilename  ; /* Name of current input file                    */
extern  int  Inhibit     ; /* Inhibit text and command processing except .{.}*/
extern  int  Itrap       ; /* Lines left to current input trap -1 if none E */
extern char  Itrap_name[]; /* Macro to invoke when Itrap reaches 0        E */
extern  int  Ismacro     ; /* 1 if Ifile is a macro,     0 if a file        */
extern  int  Isdiv       ; /* 1 if Ofile is a diversion, 0 if a file        */
extern  int  Leader      ; /* Current leader character                    E */
extern  int  Linenum     ; /* Current output line number                    */
```

```
extern  char **Macv      ; /* Macros arguments for current macro level        */
extern  int  Nobreak     ; /* Nobreak character                             E */
extern  int  Nospace     ; /* Supress spacing as per .ns command               */
extern  int  No_cntl     ; /* Don't print control characters                   */
extern  int  Nm_blanks   ; /* .nm will number blank lines too if true       E */
extern  int  Nm_on       ; /* Line numbering enabled by .nm cmd              E*/
extern  int  Nm_mult     ; /* The M argument of the most recent .nm command E*/
extern  char *Nm_str     ; /* The S argument of the most recent .nm command E*/
extern  int  Nr_cpmode   ; /* Nroff copy mode, expand \ in macro definitions */
extern  int  Num_bold    ; /* Remaining number of input lines to do bold   E */
extern  int  Num_center  ; /* Remaining number of input lines to center    E */
extern  int  Num_os      ; /* Number of input lines to print overstruck    E */
extern  int  Num_under   ; /* Remaining number of input lines to underline E */
extern  FILE *Ofile      ; /* Output file descriptor                          */
extern  int  Page_ch     ; /* Translates to page number in 3 part titles      */
extern  int  Plain       ; /* Suppress all bold, underline, and overstrike    */
extern  int  Ptab []     ; /* Proportional spacing table                      */
extern  int  Quit        ; /* Terminate nroff when set by .ex command         */
extern  int  Tab         ; /* Tab repetition character                      E */
extern  int  Tabwidth    ; /* Width of input tab stops                        */
extern  int  Tabs_enabled ; /* Expand tabs only if true                       */
extern  int  TempIn      ; /* Temporary indent column                       E */
extern  int  Title_len   ; /* 3 part title line length (set with .lt cmd)  E */
extern  int  Verbose     ; /* Echo commands to stdout as they're executed     */
extern  int  Wordstar    ; /* Wordstar-mode output                            */

extern  char *Lmarg_str  ; /* String used in .ml command                    E */
extern  char *Rmarg_str  ; /* String used in .mc command                    E */
extern  char *Bd_on      ; /* Send to printer to turn bold face on            */
extern  char *Bd_off     ; /* Send to printer to turn bold face off           */
extern  char *Ul_on      ; /* Send to printer to turn underline on            */
extern  char *Ul_off     ; /* Send to printer to turn underline off           */
extern  char *Os_on      ; /* Send to printer to turn Overstrike on           */
extern  char *Os_off     ; /* Send to printer to turn Overstrike off          */
extern  int  Bold        ; /* Boldface currently active                     E */
extern  int  Over        ; /* Overstrike currently active                   E */
extern  int  Italics     ; /* Italics currently active                      E */

extern  char *Dn_str     ; /* Send to printer to send cursor down 1/2 line    */
extern  char *Up_str     ; /* Send to printer to send cursor up 1/2 line      */
extern  int  Vs_amt      ; /* This many \u or \d cmds moves one line          */

extern  char *Left_str   ; /* Send to printer to go left          */
extern  char *Right_str  ; /* Send to printer to go right         */
extern  int  Hs_amt      ; /* Above moves 1/n spaces              */
```

End Listing Eleven

Listing Twelve

```
/*----------------------------------------------------------------------
 *      Characters are handled internally as CTYPE's rather than chars.
 *      The routines in nrmap.c copy character strings into CTYPE
 *      strings. The #defines in this file define the various attribute
 *      bits, etc., in a CTYPE:
 *
 *
 *      pad:    character is paddable, only used for spaces.
 *      lit:    character must be taken literally.
 *      width:  If 0, character takes no space in output. If 1, the
 *              character's width is in the currently active character
 *              width table.
 *      sh:     A soft hyphen preceedes this character.
 *      os:     Overstrike attribute (character is overstruck)
 *      bold:   Boldface attribute
 *      ul:     underline attribute
 *
 *       15  14   13     12      11   10    9     8   7          0
 *      +---+---+-----+-------+----+----+------+----+-----------+
 *      | 0 |   | pad | width | sh | os | bold | ul | character |
 *      +---+---+-----+-------+----+----+------+----+-----------+
 *
 *       15   14   13   12   10                0
 *      +---+----+----+----+-------------------+
 *      | 1 | vm | hm | cf | amount or font-ID |
 *      +---+----+----+----+-------------------+
 *
 *      vm:     vertical   motion
 *      hm:     horizontal motion
 *      cf:     change font
 */

typedef unsigned int  CTYPE;

#define CHR             0x00ff      /* Character mask       */
#define UNDERLINED      0x0100      /* underlined bit       */
#define BOLDFACE        0x0200      /* boldface bit         */
#define OVERSTRIKE      0x0400      /* overstrike bit       */
#define HYPHEN          0x0800      /* soft hyphen bit      */
#define WIDTH_BIT       0x1000      /* width bit            */
#define NOPAD_BIT       0x2000      /* space is paddable    */

#define MODE_BIT        0x8000      /* Selects one of:      */
#define VM_BIT          0x4000      /* Vertical motion      */
#define HM_BIT          0x2000      /* Horizontal motion    */
#define FONT_BIT        0x1000      /* Change font          */

#define SET_UL(c)       ((c) |=  UNDERLINED )   /* Underlined           */
#define IS_UL(c)        ((c) &   UNDERLINED )

#define SET_BD(c)       ((c) |=  BOLDFACE )     /* Bold                 */
#define IS_BD(c)        ((c) &   BOLDFACE )

#define SET_OS(c)       ((c) |=  OVERSTRIKE )   /* Overstrike           */
#define IS_OS(c)        ((c) &   OVERSTRIKE )

#define HYPHENATE(c)    ((c) |=  HYPHEN )       /* Soft hyphen          */
#define UNHYPHENATE(c)  ((c) &= ~HYPHEN )
#define HAS_HYPHEN(c)   ((c) &   HYPHEN )

#define CLRWIDTH(c)     ( (c) &= ~WIDTH_BIT)
#define SETWIDTH(c)     ( (c) |=  WIDTH_BIT)
```

(continued on next page)

Listing Twelve *(Listing continued, text begins on page 96.)*

```
#define HASWIDTH(c)       ( ((c) & (MODE_BIT | WIDTH_BIT)) == \
                            ((c) & (MODE_BIT | WIDTH_BIT))     )

#define CWIDTH(c)         (HASWIDTH(c) ?  Fonts[CURFONT].widths[(c)&CHR] : 0)
#define SPACE_SIZE        (               Fonts[CURFONT].widths[ ' '   ]    )

#define SETNOPAD(c)          ((c) |= NOPAD_BIT)
#define PADDABLE(c)       (! ((c) &  NOPAD_BIT) )

#define ISCHAR(c)         (((c) & MODE_BIT) == 0)

#define CHAR(c)           ((c) &  CHR )
#define ATTRIBUTES(c)     ((unsigned)(c) >> 8 )

#define TO_CTYPE(c)       ((CTYPE)(c) | WIDTH_BIT)
#define WHITE(c)          ( !((c) & MODE_BIT) && PADDABLE(c) && CHAR(c)==' ')

#define FVAL(c)           ( (int)((UCHAR)(c))               )
#define MVAL(c)           ( (int)((((int)(c)) << 4) >> 4) )

#define ISFONT(c)       ( ((c) & (MODE_BIT | FONT_BIT)) == \
                                 (MODE_BIT | FONT_BIT)      )

#define VERTICAL(c)     ( ((c) &  (MODE_BIT | VM_BIT)) == \
                                  (MODE_BIT | VM_BIT)       )

#define HORIZONTAL(c) ( ((c) &  (MODE_BIT | HM_BIT)) == \
                                (MODE_BIT | HM_BIT)       )

#define ISMOTION(c)     ( VERTICAL(c) || HORIZONTAL(c) )

#define MOTION(amt)     ( ((CTYPE)(amt) & 0xfff) | (MODE_BIT | HM_BIT))
```

End Listing Twelve

Listing Thirteen

```
/*  Length of text portion of current line.  */

#define TLEN       (LINLEN - (INDENT + Tempin))    /* in spaces */
#define U_TLEN     ( TLEN * SPACE_SIZE )           /* in units  */
```

End Listing Thirteen

Listing Fourteen

```
/*----------------------------------------------------------
 *                           NR.C
 *
 * (c) 1987, Allen I. Holub,    All rights reserved
 *
 * This module contains the nroff main() routine, and all
 * support for command line processing.
 *----------------------------------------------------------
 */

#include <stdio.h>
#include <fcntl.h>
#include <getargs.h>
#include <bitmap.h>
#include <signal.h>
#include "nr.h"


/* Variables set by command line switches. The non-static
 * variables are used in other modules. The others are used
 * by various routines in nrtext.c
 */

static  char *Pagelist = NULL;  /* Bit map used for -o option      */
static  char *Plist  = ""  ;    /* List of pages to print          */
        int  Plain   = 0   ;    /* suppress bold, underline, etc. */
        int  Stop    = 0   ;    /* Stop output every N pages       */
        int  No_cntl = 0   ;    /* Don't print any control characters
                                 * except \n. Used in nrtext.c
                                 */
        int  Verbose = 0   ;    /* echo commands as they're executed */
static  int  Fpage   = 1   ;    /* Number of the first page. We can't
                                 * use the PAGE number register
                                 * because number registers don't
                                 * exist yet.
                                 */
static  int  Even        = 0;   /* Print only even pages          */
static  int  Odd         = 0;   /* Print only odd pages           */
static  int  Unbuf       = 0;   /* Don't buffer the input stream */
extern  int  do_mfile() ;       /* Defined below, processes -m    */
extern  int  do_tstr()  ;       /* Defined below, processes -t    */
extern  int  do_rreg()  ;       /* Defined below, processes -r    */


static ARG  Argtab[] =
{
    {'c', BOOLEAN,          &No_cntl, "don't print (C)ontrol characters"      },
    {'d', BOOLEAN,          &Odd,     "print only o(D)d pages"                },
    {'e', BOOLEAN,          &Even,    "print only (E)ven pages"               },
    {'m', PROC,     (int *) do_mfile, "prepend (M)acro: /lib/tmac/<str>.mac" },
    {'n', INTEGER,          &Fpage,   "(N)umber first page N"                 },
    {'o', STRING,   (int *) &Plist,   "print (O)nly pages in list (<str>)"    },
    {'p', BOOLEAN,          &Plain,   "Suppress bold, underline, overstrike" },
    {'r', PROC,     (int *) do_rreg,  "set number (R)eg: -rx<num> -r(xx<num>"},
    {'s', INTEGER,          &Stop,    "(S)top every n pages"                  },
    {'t', PROC,     (int *) do_tstr,  "set s(T)ring: -tx<str> -t(xx<str>"     },
    {'u', BOOLEAN,          &Unbuf,   "Don't buffer input for debugging"      },
    {'v', BOOLEAN,          &Verbose, "(V)erbose mode, echo input commands"  }
};
```

```
/*----------------------------------------------------------*/

#define ISODD(x)        ((x) & 1)
#define ISEVEN(x)       (!ISODD(x))
#define NAMESIZE 50

            /* Full path name of macro file specified with
             * the -m command line argument. The %s will
             * be replaced with the string following the
             * -m on the command line. NAMESIZE should
             * agree with the precision field. (%1.32s).
             * A NAMESIZE sized buffer is used to hold
             * the expanded macro file name.
             */

#define MACFILE  "\\lib\\tmac\\%1.8s.mac"

/*----------------------------------------------------------*/

get_pglist( )
{
    /* Produces a page list from str. A page list is a
     * bit map with one bit representing each legal page.
     * 0 < N <= 512. Str is a null terminated string giving
     * the legal pages seperated by commas. The notation
     * "N-M" may be used to print all pages between N and M.
     * "-N" means from the beginning of the document to
     * page N. "N-" means from page N to the end of the
     * document. The special forms: -oe and -oo print all
     * even pages and all odd pages respectively.
     */

    register int        start, end;

    if( !(Pagelist = (char *) makebitmap( MAXPAGE )) )
    {
        err("Not enough memory to make page list\n");
        return;
    }

    if( (Even || Odd) && !*Plist )
        Plist = "-" ;

    while ( *Plist )
    {
        start = ( *Plist == '-') ? 1 : stoi( &Plist ) ;
        /* At this point start will be set to 1 if there
         * was a leading dash or to the number if no
         * leading dash. "Plist" will be pointing past the
         * number. If *Plist is a dash we are doing a range,
         * else we are setting a single page.
         */

        if( *Plist != '-' )
            end = start;
        else
        {
            Plist++;                    /* Skip the '-'     */
            if( !(end = stoi(&Plist)))
                end = MAXPAGE;          /* No # following - */
        }

        for(; start <= end; start++ )
        {
            if
            (    ( !Even && !Odd                   )
               ||( Even  && ISEVEN(start) )
               ||( Odd   && ISODD (start) )
            )
            {
                setbit( start, Pagelist, 1 );
            }
        }

        while( *Plist == ',' )
            Plist++;
    }
#ifdef DEBUG
    printf("Only the following pages will be printed: \n");

    for( start = 1; start <= MAXPAGE; start++ )
        if( testbit(start, Pagelist) )
            printf("%d, ", start );
    printf("\n");
#endif
}

/*----------------------------------------------------------*/

ispage( n )
{
        /*  Return 1 if n is a legal page to print (is in
         *  the page list or no -o option was ever given).
         */

        return( Pagelist ? testbit(n, Pagelist) : 1 );
}

/*----------------------------------------------------------*/

do_mfile( name )
char    *name;
{
    /* This routine is called when the -m<str> argument
     * is encountered on the command line. It is called
     * directly by getargs.
     */

    char    nbuf[NAMESIZE]    ; /*  Macro file name  */

    if( *name )
```

(continued on next page)

Listing Fourteen *(Listing continued, text begins on page 96.)*

```
    {
            INLINES = 0;
            sprintf(nbuf, MACFILE, name);
            Ifilename = nbuf ;

            if( !(  Ifile = fopen(nbuf, "r")    ))
            {
                err("Can't open macro file: %s\n", nbuf );
                exit( 1 );
            }

            process( Ifile, Ifilename, 0, 0 );

            fclose( Ifile );
    }
}

/*------------------------------------------------------------*/

do_tstr( str )
char    *str;
{
    /* Called by getargs() when -t is encountered.
     * Given -tx<str> or -t(xx<str> initializes
     * register x or xx to <str>.
     */

    static  char    line[ MAXSTR ];
    char    name[4];
    extern  sgetc();

    name[2] = name[1] = 0;

    if( *str )
    {
            if( (name[0] = *str++)  == '(' )
            {
                    if( *str && *(str + 1) )
                    {
                            name[0] = *str++;
                            name[1] = *str++;

                    }
            }

            Ifile = (FILE *)&str;
            getline( line, 0, sgetc );
            ds(name, line );

            return;
    }

    fprintf(stderr, "Illegal string name on command line");
}

/*------------------------------------------------------------*/

do_rreg(str)
char    *str;
{
    /*  Processes -r command line argument. Given
     *  -rx<str> or -r(xx<str> initializes number
     *  register x or xx to <str>.
     */

    char name[4];

    name[2] = name[1] = 0;

    if( *str )
    {
            if( (name[0] = *str++)  == '(' )
            {
                    if( *str && *(str + 1) )
                    {
                            name[0] = *str++;
                            name[1] = *str++;
                    }
            }

            putnreg(name, 0, atoi(str) , 0, 1, 0);
            return;
    }

    fprintf(stderr,"Illegal register name on command line");
}

/*------------------------------------------------------------*/

onintr()
{
        /* Treat a Ctrl-C or Ctrl-Break as if we've
         * executed a .ex command.
         */

        signal( SIGINT, onintr );
        mac_clean();    /* Delete all macro disk files */
        exit( 1 );
}

/*------------------------------------------------------------*/

usage()
{
        exit( 1 );
}

/*------------------------------------------------------------*/

main(argc, argv)
int     argc;
char    **argv;
{
                                /* Initialize:                    */
        df("R", "");            /* default, monospaced, font      */
        init_text();            /* text module                    */
        init_nreg();            /* pre-defined number registers   */

        signal( SIGINT, onintr );  /* Treat ^C like .ex */

        argc = getargs(argc, argv, Argtab,
                        sizeof(Argtab)/sizeof(ARG), usage);


        PAGE = Fpage;   /*  Number of first page of
                         *  document as per -n argument.
                         */

        if( *Plist || Even || Odd )
                get_pglist();

        Ofile = stdout;

        do {    /* process a single input file */
                INLINES = 0;

                if( argc <= 1 )
                {
                        Ifilename = ""    ;
                        Ifile     = stdin ;
                }
                else
                {
                    Ifilename = *++argv ;
                    Ifile = fopen( Ifilename, "r" );
                    if( !Ifile )
                    {
                        err("Can't open input file <%s>\n",
                                                Ifilename);
                        break;
                    }
                }

                /* The setvbuf call puts us into unbuffered
                 * input mode.
                 */

                if( Unbuf )
                    setvbuf( Ifile, NULL, _IONBF, 0 );

                process( Ifile, Ifilename, 0, 0 );
                fclose ( Ifile                  );

        } while( --argc > 1  &&  !Quit );

        brk();          /* Flush output buffer      */

        /* Do the end macro if there is one. If we
         * don't clear Quit before expanding the macro,
         * getline() will return end of file and the macro
         * won't be executed.
         */

        if( *Endm )
        {
            Quit = 0;
            expand_macro(Endm);
        }

        mac_clean();    /* Delete all macros disk files */
        exit( 0 );
}
```

End Listing Fourteen

Listing Fifteen

```
/*
 * NRCMD.C
 *
 * Copyright (c) 1987,  Allen I. Holub. All rights reserved.
 *
 * This module contatins the routines to process individual
 * commands. Rotines are accessed via the Cmdtab (see nr.c
 * and below
 */

#include <stdio.h>
#include <ctype.h>
#include "nr.h"

extern CTAB     Cmdtab[] ;
extern int      Ctabsize ;
extern char     *skipto(), *skipspace();  /* in tools.lib */
extern char     *bsearch() ;

/*------------------------------------------------------------*/

cmdcmp( matchstr, cp )
register unsigned char   *matchstr;
register CTAB            *cp;
{
        /* Comparison routine used by search called in
         * command() below.
         */
```

```
        register unsigned int   l, r;

        D( printf("comparing %2.2s ", matchstr) );
        D( printf("and %2.2s, "     , cp->cmd ) );

        l = matchstr[0];
        r = (cp->cmd)[0];

        if( l == r )
        {
                l = matchstr[1] ;
                r = (cp->cmd)[1];

                if( isspace(l) )
                        l = 0 ;
        }

        D( printf("returning %d\n", l - r) );

        return( l - r );
}

/*----------------------------------------------------------*/

int     numarg(s, offset)
char    **s;
int     *offset;
{
        /* Get value of a numeric argument from *s. If the
         * number is followed by an i, inches are converted
         * to spaces. If the number is preceeded by a '+' or
         * a '-' offset is set to 1. The argument may be an
         * expression and spaces are ignored. However the
         * argument must have been enclosed in double quotes
         * for a space to be part of the argument. S is
         * advanced past the numeric componant and any
         * trailing whitespace. Return the value of the
         * argument.
         */

        int             error;
        extern int      getvar(), null();
        extern double   parse();
        double          val = 0.0 ;

        if( **s )
        {
                if ( **s == '+' )
                {
                        *offset = 1;
                        (*s)++;
                }
                else if( **s == '-' )
                        *offset = 1;

                val = parse( s );
        }

        return( (int)val );
}

/*----------------------------------------------------------*/

splitfields( cur, next )
char    **cur, **next;
{
        /*
         * Split cur into two fields. Modify next to point
         * at the beginning of the second or at end of line.
         *
         * Leading and trailing white space around the first
         * field is skipped quoted arguments are recognized
         * as being a single field, even if the quoted
         * string contains whitespace.
         */

        register char   *p;

        p = *cur;
        p = skipspace(p, Esc);

        if( *p == '"' )
        {
                *cur = ++p;
                p = skipto('"', p, Esc) ;
        }
        else
        {
                *cur = p ;
                p = skipto(' ', p, Esc );
        }

        if( *p )
                *p++ = '\0';        /* Terminate current field */

        p = skipspace( p, Esc ); /* Skip to next field       */
        if( *p == '"' )          /* strip any quotes         */
        {
                *skipto('"', ++p, Esc ) = 0;
        }

        *next = p;
}

/*----------------------------------------------------------*/

command( first )
char    *first;
```

(continued on next page)

C CHEST

Listing Fifteen

(Listing continued, text begins on page 96.)

```
{
    /* Process a command found in "first". Do this by finding
     * the command in Cmdtab. If the command is found then
     * the associated subroutine (in the Cmdtab) is executed.
     *
     * The calling convention depends on the command type.
     * There are 4 types:
     *
     * type 0:           .xx <optional string>
     * type 1:           .xx <number> <optional string>
     * type 2:           .xx <string> <number> [optional tail]
     * type 3:           .xx <string> <optional string>
     *
     * A <string> is passed null terminated with leading and
     * trailing white space or quotes stripped. A <number>
     * is passed as an int. An <optional string> is passed
     * without trailing white space or leading and trailing
     * quotes stripped.
     *
     * Type 0: (* action)( str, dobreak );
     *                 char    *str;
     *
     * Type 1: (* action)( val, str, offset, dobreak      );
     * Type 2: (* action)( val, str, offset, dobreak, tail);
     *                 double  val;
     *                 char    *str;
     *                 char    *tail;
     *
     * Type 3:     (* action)( leftstr, rightstr, dobreak );
     *                 char    *leftstr, *rightstr ;
     *
     * Note that command() will mess up the input string,
     * putting a null after the first character. If you need
     * to keep the string around for longer than one command,
     * copy it somwhere safe.
     */

    extern   CTAB *search(); /* routine to search for cmd */
    register CTAB *cmd     ; /* current command           */
    char          *second  ; /* points at second argument */
    char          *p       ; /* general-purpose pointer   */
    int           val    =0; /* value of numeric argument */
    int           offset =0; /* true if val is an offset  */
    int           rval   =0; /* return value              */
    int           dobreak  ; /* True if a normal Cmd char,
                              * false if a nobreak command
                              * char.
                              */

    dobreak = (*first++ == Cmd_chr);

    while( *first  &&  (*first & 0x7f) <= ' ' )
            first++;

    if( !*first )                   /* This is a comment line */
            return 0;

    cmd = (CTAB *) bsearch(first, Cmdtab, Ctabsize,
                                      sizeof(CTAB), cmdcmp);

    if( !cmd )
    {
        /*  Command isn't in the table. See if it's a
         *  macro. Print an error message if it isn't.
         */

        if( !expand_macro( first ) )
            err(".%2s not a command or macro\n", first );

        return 0;
    }

    if( Inhibit && cmd->inhib ) /* Input is inhibited. See */
            return 0;           /* See doif() in nrmsc.c   */
                                /* for details.            */

    offset   =  0       ;       /* advance past the actual */
    first    += 2       ;       /* command.                */

    if( cmd->type == 0 )
    {
            first = *first ? skipspace(first,Esc) : cmd->def;

            if( *first == '"' )
                    *skipto('"', ++first, Esc) = 0;

            rval = ( *(cmd->action))( first, dobreak );
            goto exit;
    }

    splitfields( &first, &second );


    switch( cmd->type )
    {
    case 1:
            p = *first ? first : cmd->def ;

            val  = numarg( &p , &offset );
            rval = (*(cmd->action))( val, second, offset,
                                                    dobreak);

            break;

    case 2:
            p = *second ? second : cmd->def ;
```

```
            val = numarg( &p, &offset );

            p = skipspace(p, Esc);
            if( *p == '\"' )
                    *skipto('\"', ++p, Esc) = '\0';

            rval = (*(cmd->action))( val, first, offset,
                                                dobreak, p );
            break;

    case 3:
            rval = (*(cmd->action))( *first? first: cmd->def,
                                            second, dobreak);
            break;

    default:
            err("** Internal Error, bad type: %d in Cmdtab\n",
                    cmd->type );
            break;
    }

exit:
    return( cmd->inhib ? 0: rval );
}
```

End Listing Fifteen

Listing Sixteen

```
#include <stdio.h>
#include <hash.h>
#include "nr.h"

/*----------------------------------------------------------
 * NREG.C
 *
 * Copyright (c) 1987,  Allen I. Holub. All rights reserved.
 *
 * This module holds routines for mainpulating and accessing
 * number registers.
 *
 *----------------------------------------------------------
 */

static int Regnum = 0;  /* Used to print number registers */

HASH_TAB  *Nregs = 0;   /* Hash table that holds number    */
                        /* registers.                       */

/*----------------------------------------------------------*/

init_nreg()
{
    extern NREG *putnreg();
    int         garbage;

    Nregs = maketab( 127 );

    Nrpg      = putnreg( "%" , ARABIC,   DEF_PAGE,    0,1,1);
    Nrargs    = putnreg( ".$", READONLY, DEF_NARGS,   0,1,1);
    Nrlines   = putnreg( ".c", READONLY, DEF_INLINES, 0,1,1);
    Nrvplace  = putnreg( ".d", READONLY, DEF_VERT,    0,1,1);
    Nrfont    = putnreg( ".f", READONLY, DEF_FONT,    0,1,1);
    Nrindent  = putnreg( ".i", READONLY, DEF_INDENT,  0,1,1);
    Nrllen    = putnreg( ".l", READONLY, DEF_LINLEN,  0,1,1);
    Nrtlen    = putnreg( ".n", READONLY, DEF_TEXTLEN, 0,1,1);
    Nroffset  = putnreg( ".o", READONLY, DEF_OFFSET,  0,1,1);
    Nrplen    = putnreg( ".p", READONLY, DEF_PGLEN,   0,1,1);
    Nrtotrap  = putnreg( ".t", READONLY, DEF_TOTRAP,  0,1,1);
    Nrfill    = putnreg( ".u", READONLY, DEF_FILL,    0,1,1);
    Nrv       = putnreg( ".v", READONLY, DEF_LS,      0,1,1);
    Nrdl      = putnreg( "dl", ARABIC,   DEF_WIDTH,   0,1,1);
    Nrdn      = putnreg( "dn", ARABIC,   DEF_HEIGHT,  0,1,1);
    Nrdy      = putnreg( "dy", ARABIC,   DEF_DAY,     0,1,1);
    Nrh       = putnreg( "h",  ARABIC,   0,           0,1,1);
    Nrhp      = putnreg( "hp", ARABIC,   DEF_HORIZ,   0,1,1);
    Nrln      = putnreg( "ln", ARABIC,   DEF_LINE,    0,1,1);
    Nrnl      = putnreg( "nl", READONLY, 1,           0,1,1);
    Nrm       = putnreg( "m",  ARABIC,   0,           0,1,1);
    Nrmo      = putnreg( "mo", ARABIC,   DEF_MONTH,   0,1,1);
    Nrs       = putnreg( "s",  ARABIC,   0,           0,1,1);
    Nrdw      = putnreg( "wd", ARABIC,   0,           0,1,1);
    Nryr      = putnreg( "yr", ARABIC,   DEF_YEAR,    0,1,1);

    time( &HOUR,  &MIN, &SEC,  &garbage );
    date( &MONTH, &DAY, &YEAR, &WEEKDAY );

    WEEKDAY++;  /* Translate 0-6 to 1-7 for compatability */
}

/*----------------------------------------------------------*/

not_deletable( name )
char    *name;
{
        /* Return true if name is a pre-defined but not read
         * only number register.
         */

        register int    c1, c2;

        c1 = name[0];
        c2 = name[1];

        return( ( c1 == '%'  &&  c2 == 0 )   ||
                ( c1 == 'd'  &&  c2 == 'h')  ||
                ( c1 == 'd'  &&  c2 == 'l')  ||
```

(continued on next page)

Listing Sixteen *(Listing continued, text begins on page 96.)*

```
                ( c1 == 'd'  &&  c2 == 'y')  ||
                ( c1 == 'h'  &&  c2 == 'p')  ||
                ( c1 == 'l'  &&  c2 == 'n')  ||
                ( c1 == 'm'  &&  c2 == 'o')  ||
                ( c1 == 'y'  &&  c2 == 'r')  );
}

/*------------------------------------------------------------*/

NREG    *putnreg(name, fmt, val, offset, create, incr_amt)
char    *name;
int     fmt, val, offset;
{
    /* Change the value of any field of a number
     * register called "name." If the number register
     * does not exist and "create" is true, then create
     * it. Offset is treated as follows:
     *
     *      offset <  0  number register not modified
     *      offset == 0  number register  = val
     *      offset >  0  number register += val
     *
     * If fmt is non-zero put it into the format field,
     * else leave the nfmt field alone. Ditto with
     * incr_amt. Number registers who's format is
     * READONLY can't be modified. Return a pointer to
     * the register if it was found or created or
     * NULL if register not found and/or not created.
     */

    register NREG       *pnode;

    if ( *name == '\0')
            return NULL;

    if( !(pnode = (NREG *) findsym( Nregs, name )) )
    {
        if( !create )
        {
            err("Number register doesn't exist\n");
            return NULL;
        }

        pnode = (NREG *) addsym( Nregs, name, sizeof(NREG));
        pnode->nfmt      = ARABIC  ;
        pnode->nval      = 0       ;
        pnode->incr_amt  = 1       ;
    }

    if( pnode->nfmt == READONLY )
    {
        err("Can't modify a read/only number register\n");
        return (NREG *) 0;
    }

    if( incr_amt )
            pnode->incr_amt = incr_amt ;

    if(fmt)
            pnode->nfmt = fmt;

    if( offset >= 0 )
    {
            if( offset )
                    pnode->nval += val  ;
            else
                    pnode->nval  = val  ;
    }

    return(pnode);
}

/*------------------------------------------------------------*/

rm_nreg(name)
char    *name;
{
    /* Remove number register "name" if it exists.
     */

    register NREG       *node;

    if( !(node = (NREG *) findsym( Nregs, name )) )
        err("Can't find number register <%1.2s>\n", name );

    else if(  node->nfmt == READONLY || not_deletable(name))
        err("May not delete pre-defined number register\n");

    else
        delsym( Nregs, (BUCKET *) node );
}

/*------------------------------------------------------------*/

prnt( name, p )
NREG    *p;
{
    printf("%2s = %4d (format = %c, incr by %d)",
                        name, p->nval, p->nfmt, p->incr_amt);
```

```
    printf( (++Regnum % 2) ? "\t|  " : "\r\n" );
}

pr_nregs()
{
    /* Print out the values of all the number registers
     */

    Regnum = 0;
    ptab( Nregs, prnt );
    printf("\r\nThere are %d number registers\r\n", Regnum);
}

/*----------------------------------------------------------*/

int     nrtoi( p, fmt )
char    *p ;
int     *fmt ;
{
        /* Return the value of the number reg. variable
         * whose name is in string. Fmt is modified to be
         * the contents of the registers format field.
         * If *p is a +, the number register is
         * auto pre-incremented, if it's a -, it's pre-
         * decremented. If it's 0, it isn't modified.
         * Non-existant number registers evaluate to 0.
         */

        int     rval = 0 ;
        int     i    = 0 ;
        NREG    *node ;

        *fmt = ARABIC;      /* Default error return values */

        if( !*p )

            err("Missing number register name\n");
        else
        {
            if( *p == '-' || *p == '+' )
                i = (*p++ == '+') ? 1 : -1 ;

            if( node = (NREG *) findsym(Nregs, p) )
            {
                *fmt        = node->nfmt ;
                node->nval += ( node->incr_amt * i );
                rval        = node->nval ;
            }
        }

        return( rval );
  }
```

End Listing Sixteen

Listing Seventeen

```
/* NRGLBLS.C:   Global variables used by several modules
 *
 * Copyright (c) 1987,  Allen I. Holub.
 *
 * Global variables used by the various nroff routines
 *----------------------------------------------------------
 */

#include <stdio.h>
#include "nr.h"

/*----------------------------------------------------------
 *      Nodes for pre-defined number registers
 */

NREG    *Nrpg     ;
NREG    *Nrargs   ;
NREG    *Nrlines  ;
NREG    *Nrvplace ;
NREG    *Nrfont   ;
NREG    *Nrindent ;
NREG    *Nrllen   ;
NREG    *Nrtlen   ;
NREG    *Nroffset ;
NREG    *Nrplen   ;
NREG    *Nrtotrap ;
NREG    *Nrfill   ;
NREG    *Nrdl     ;
NREG    *Nrdn     ;
NREG    *Nrdy     ;
NREG    *Nrhp     ;
NREG    *Nrln     ;
NREG    *Nrmo     ;
NREG    *Nrnl     ;
NREG    *Nryr     ;
NREG    *Nrdw     ;
NREG    *Nrh      ;
NREG    *Nrm      ;
NREG    *Nrs      ;
NREG    *Nrv      ;

/*----------------------------------------------------------
 * Tabstop is an array of tabstops, indexed by column number.
 * 0 means no tab stop in that column, 'R' means right
 * adjusting, 'L' is left adjusting, 'C' is centering. Tab
 * positions are all increments of spaces from the left
 * margin. The leftmost column is column 1. tabs are set
 * and cleared by tabset() and tabclr() (in nrmsc.c). They
 * are used in nrtext.c
 */
```

(continued on next page)

Listing Seventeen *(Listing continued, text begins on page 96.)*

```
TSTOP      Tabstop =
{
           0,
           0,  0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0,
          'L', 0, 0, 0,  0, 0, 0, 0
};

/*----------------------------------------------------------*/

FONT Fonts[NUMFONTS]; /* Font table. Fonts[0] defines the */
                      /* default, non-proportional font   */
                      /* attached to the name R, it is    */
                      /* created in main().               */

/*----------------------------------------------------------
 *      Msc. other Global variables
 */

int  Adjmode    = BOTH ; /* Current adjustment mode                          */
int  Adjusting  = 0    ; /* One if adjusting lines. We can't just use an
                          * adjustment mode here because we have to remember
                          * the most recent adjustment mode.
                          */
int  Cmd_chr    = '.'  ; /* Command character                                */
int  Cont_ul    = 0    ; /* Num of input lines to continuously underline    */
int  Divtrap    = -1   ; /* Distance to diversion trap (-1 if none)         */
int  Divwidth   = 0    ; /* Width of current diversion. This differs from   */
                         /* WIDTH, which is the width of the most-recent    */
                         /* diversion.                                       */
char *Endm      = ""   ; /* Name of macro invoked at end of input           */
int  Esc        = '\\' ; /* Current escape character                        */
int  Hyphen_chr = '%'  ; /* Soft hyphen is \<Hyphen_chr>. default = \%      */
int  Hyphenate  = 0    ; /* Hyphenation is enabled during filling only      */
int  Itrap      = -1   ; /* Distance to input line trap, -1 if none         */
int  Isdiv      = 0    ; /* 1 if Ofile is diversion, 0 if file              */
int  Ismacro    = 0    ; /* 1 if Ifile is a macro,   0 if a file            */
int  Leader     = '.'  ; /* Current leader character                        */
char **Macv     = 0    ; /* Macro arguments for current macro level         */
int  Nobreak    = '`'  ; /* Nobreak character                               */
int  Nospace    = 0    ; /* Inhibit spacing as per .ns command              */
int  Nr_cpmode  = 0    ; /* Nroff copy mode, expand \ in macro definitions*/

int  Num_bold   = 0    ; /* Remaining number of input lines to print bold */
int  Num_center = 0    ; /* Remaining number of input lines to center     */
int  Num_os     = 0    ; /* Number of input lines to overstrike           */
int  Num_under  = 0    ; /* Remaining number of input lines to underline  */
int  Page_ch    = '%'  ; /* Translates to page number in 3 part titles    */
int  Quit       = 0    ; /* Terminate nroff when set by .ex command       */
int  Tab        = ' '  ; /* Tab repetition character                      */
int  Tabwidth   = 8    ; /* Width of input tab stops                      */
int  Tabs_enabled = 1  ; /* Expand tabs only if true                      */
int  TempIn     = 0    ; /* Temporary indent column, (offset from INDENT) */
int  Title_len  = 80   ; /* 3 part title line length (set with .lt cmd)   */
int  Wordstar   = 0    ; /* Wordstar-mode output                          */
int  H_units    = 120  ; /* Number of horizontal units in an inch         */
int  H_space    = 10   ; /* Number of horizontal units in a space         */

int  Bold       = 0    ; /* Boldface is currently active                  */
int  Over       = 0    ; /* Overstrike is currently active                */
int  Italics    = 0    ; /* Italics is currently active                   */
char *Bd_on     = ""   ; /* Send to printer to turn boldface on           */
char *Bd_off    = ""   ; /* Send to printer to turn boldface off          */
char *Ul_on     = ""   ; /* Send to printer to turn underline on          */
char *Ul_off    = ""   ; /* Send to printer to turn underline off         */
char *Os_on     = ""   ; /* Send to printer to turn overstrike on         */
char *Os_off    = ""   ; /* Send to printer to turn overstrike off        */

char *Dn_str    = ""   ; /* Send to printer to send cursor down 1/2 line */
char *Up_str    = ""   ; /* Send to printer to send cursor up 1/2 line   */
int  Vs_amt     =  1   ; /* This many \u or \d cmds moves one line       */

char *Left_str  = "\b"; /* Send to printer to go left                     */
char *Right_str = " " ; /* Send to printer to go right                    */
int  Hs_amt     =  1  ; /* Above moves 1/n spaces                         */

char *Lmarg_str = ""  ; /* String used in .ml command                     */
char *Rmarg_str = ""  ; /* String used in .mc command                     */

char Dtrap_name[3] = {0,0,0} ;      /* Macro executed when Divtrap reached */
char Itrap_name[3] = {0,0,0};       /* Macro executed when Intrap == 0     */

FILE *Ifile     = stdin;            /* Current input file                  */
FILE *Ofile     = stdout;           /* Output file descriptor              */
char *Ifilename = "standard input"; /* name of input file                  */

int  Nestlev    = 0  ;  /* Nesting level determined by .{/.}               */

int  Nm_on      = 0  ;  /* Line numbering enabled by .nm cmd               */
int  Nm_blanks  = 0  ;  /* If true then blank lines are numbered by .nm   */
int  Nm_mult    = 1  ;  /* The M argument of the most recent .nm command */
char *Nm_str    = " ";  /* The S argument of the most recent .nm command */
int  InhIbit       = 0; /* Inhibit all text and command processing except
                         * .} and .{
                         */
```

End Listing Seventeen

Listing Eighteen

```c
/*------------------------------------------------------------
 * NRINP.C: Input and escape sequence processing. Also
 *          contains process(), the highest level input
 *          processing routine.
 *
 * (c) 1987, Allen I. Holub,   All rights reserved
 *
 *------------------------------------------------------------
 */

#include <stdio.h>
#include <ctype.h>
#include "nr.h"

#define ishex(c) (('0'<=(c) && (c)<='9') || \
                  ('A'<=(c) && (c)<='F') )

#define tohex(c) (('0'<=(c) && (c)<='9') ? (c)-'0' \
                                      : ((c)-'A') + 0x0a)

/*------------------------------------------------------------*/
static int Abort_process = 0;   /* Used by escape to tell
                                 * process to abort the
                                 * current process.
                                 */

static int  New_font = 0;               /* Used by chfont() */

extern char *expandstr(char*, char*, int);  /* nrmac.c    */
extern char *cpy( char*, char* );           /* tools.lib  */

/*------------------------------------------------------------*/

gnum( inp, ifile, nextc )
int     (*inp)(), *nextc ;
FILE    *ifile;
{
    /* Get a decimal number from input, the number can be
     * given explicitly or as a number register
     * (ie. \l\n(xx- is legal, it will draw as many '-'s
     * as are specified in the \n(xx number register. The
     * number is returned and *c is modified to hold the
     * first nondigit.
     */

    int c, i, sign = 1 ;
    UCHAR       name[4];

    if( (c = (* inp)(ifile))  == Esc )
    {
        if( (c = (* inp)(ifile))  == 'n' )
        {
            gname( name, inp, ifile, 1 );
            i = nrtoi( name, &c );
            c = (* inp)(ifile)   ;
        }
        else
            err("Must use number or number register\n");
    }
    else
    {
        if( c == '-' )
        {
            sign = -1 ;
            c = (*inp)(ifile);
        }
        else if ( c == '+' )
            c = (*inp)(ifile);

        for( i = 0 ; isdigit(c) ; c = (*inp)(ifile) )
        {
            i = (i * 10) + (c - '0') ;
        }
    }

    *nextc = c;
    return i * sign ;
}

/*-----------------------------------------------------------*

gname( name, inp, ifile, nreg )
char    *name;
int     (*inp)();
FILE    *ifile;
{
    /* Get a string or number register name from inp and
     * put it into name. In the case of an autoincrement
     * ( \n-(xx for example ), a leading - or + is put
     * into the name too.
     */

    register int        c;

    c = (*inp)(ifile) ;

    if( nreg && (c == '+' || c == '-') )
    {
        *name++ = c ;
        c = (*inp)(ifile) ;
    }

    if( c == '(' )
    {
        *name++ = (*inp)(ifile);
        *name++ = (*inp)(ifile);
```

(continued on next page)

Listing Eighteen *(Listing continued, text begins on page 96.)*

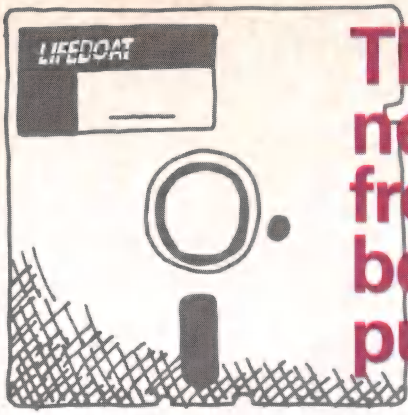
```
    }
    else
        *name++ = c ;

    *name   = 0 ;
}

/*------------------------------------------------------------*/

#define get_quote(str) if( (*inp)(ifile) != '\'' )           \
        {                                                     \
                err("Missing quote in %s\n", str );           \
                break;                                        \
        }                                                     \
        else

/*------------------------------------------------------------*/

int     escape( tstart, target, copymode, inp, ifile, maxch)
UCHAR   *tstart, **target;
int     (*inp)();
FILE    *ifile;
{
    /* {
     * Expand escape sequences, using inp() to get
     * additional input when required. Expand at most maxch
     * characters. Target is modified to point past the
     * expanded string. The input character following the
     * escape sequence is returned.
     *
     * Tstart is the array into which characters go. *target
     * is the current location in that array. The input
     * character following the escape sequence is returned.
     *
     * Note that the string "\t" will actually put a tab
     * character (unexpanded) into the input stream. An ASCII
     * ^I or a \T, will have been expanded by getline().
     *
     * Copy mode is a subset of normal mode used for macro
     * definitions. In normal copy mode (Nr_cpmode == 0) the
     * only recognized escape sequences are \" and
     * \<newline>. Other sequences are just copied to the
     * target string. In nroff-compatible copy mode
     * (Nr_cpmode != 0), the following are recognized:
     *          \. \\ \* \$ \n \" \<newline>
     *
     * Nested \* expansions are supported and the strings
     * can contain other escape sequences (like \nx). Note
     * that nesting is handled recursively in that
     * expandstr(), called below, will call escape() to
     * expand internal escape sequences. For reasons of
     * nroff compatability \{ is mapped to .{ and \} will
     * cause process() to terminate immediatly after doing
     * the line on which the \} was found, as if it had
     * seen a .}
     */

/*------------------------------------------------------------*/
    register int i;         /* temporary                    */
    int         c;          /* current input character      */
    int         j;          /* temporary                    */
    int         linechar;   /* line-drawing character       */
    UCHAR       *bp;        /* general-purpouse pointer     */
    UCHAR       *dest;      /* Pointer to target array      */
    UCHAR       name[8];    /* string or number reg name    */
    static UCHAR temp[80];  /* buffer used by itoascii()    */
                            /* to translate number          */

#ifdef DEBUG
    printf("escape:targ=0x%x, cpymode=%d, inp=0x%x, maxch=%d",
                            target, copymode, inp, maxch );
    printf("\nescape: macv = 0x%x\n", Macv);
    for( i=10 ; Macv && --i>=0 ; )
            printf("escape: %2d: <%2.2s>\n", i, Macv[i] );
#endif

    dest = *target;

    /* Cases in the following switch are expanded whether
     * or not we're in copy mode. "c" holds the character
     * following the escape character.
     */

    switch( c = (*inp)(ifile) )
    {
    case '\"':

        /* Throw away input up  to the newline or end of
         * file. Then delete all white  space preceeding the
         * comment. Use "goto exit" in  order to avoid
         * getting another input character.
         */

        while(  (c = (*inp)(ifile)) != '\n'  &&  c != EOF )
                ;

        while(  *--dest == ' ' || *dest == '\t' )
                if(  dest < tstart )
                        break;
        ++dest  ;
        goto exit ;


    case '\n':       /* line continuation, just eat the \n */
        goto newchar;

    default:
        if( copymode && !Nr_cpmode )
        {
                /* In   non-nroff copy mode, only \" and \<CR>
                 * are recognized. Everything   else goes
                 * through to   the output.
                 */

                *dest++ = Esc   ;
                *dest++ = c     ;
                goto newchar;
        }
    }

    /* Cases in the following switch are expanded either in
     * nroff-compatible copy mode or when not in copy mode
     * of any sort (because of the goto branch in the default
     * case of the previous switch()). They are not expanded
     * in normal copy mode.
     */

    switch( c )
    {
    case '.':
    case '`':

        *dest++ = LITCHAR ;
        *dest++ = c;
        HORIZ++;
        goto newchar;

    case '$':                           /* \$N  1 <= N <= 9 */

        /* Expand macro arguments. The  leftmost one is in
         * Macv[0] but, for nroff comapatability we access
         * it as \$1. \$0 can not be accessed.
         */

        if( !Macv )
        {
            err("\\$<num> can only be used in a macro\n" );
            goto newchar;
        }

        if( (i  = (*inp)(ifile) - '0') < 1  ||  i > 9 )
        {
            err("\\$n:  invalid number, 1 <= n <= 9\n");
            goto newchar;
        }

        for( bp = Macv[i-1] ; *bp  &&   --maxch >= 0 ; )
        {
            HORIZ++ ;
            *dest++ = *bp++;
        }
        goto newchar;


    case 'n' :            /* \nx  or  \n(xx                 */

        gname(  name, inp, ifile, 1 );
        i = nrtoi( name, &j ) ;

        if( j == READONLY )
                j = ARABIC;

        i = itoascii(temp, j, i);

        if( maxch < i )
            err("Buffer too small to expand register\n");
        else
        {
                dest = cpy( dest, temp );
                HORIZ   += i;
        }

        goto newchar;

    case '*' :                  /*    \*(xx  or \*x     */

        gname(  name, inp, ifile, 0 );
        bp    =  dest ;
        dest  =  expandstr( name, dest, maxch );
        HORIZ += dest - bp ;
        goto newchar;

    default:

        if( copymode )         /* We're in nroff-compatable  */
        {                      /* copy mode                  */
                if( c   != Esc )
                    *dest++ =   Esc ;

                *dest++ = c     ;
                goto newchar;
        }
        break;
    }

    /* Cases in the following switch are expanded only
     * when we're not in copy mode of any sort.
     */


    switch ( c )
    {
    case ' ' : *dest++ = UP_SPACE;  HORIZ++;         break;
    case '0' : *dest++ = ' T;       HORIZ++;         break;
    case '|' : /* ignored */                         break;
    case '^' : /* ignored */                         break;
```

(continued on page 68)

Listing Eighteen *(Listing continued, text begins on page 96.)*

```
    case 'N' : *dest++ = '\n';                         break;
    case '-' : *dest++ = '-';       HORIZ++;           break;
    case '&' : *dest++ = LITCHAR;                      break;
    case 'z' : *dest++ = ZWIDTH;                       break;

    case 't' : *dest++ = LITCHAR ; /*fall through*/
    case 'T' : *dest++ = '\t';                         break;

    case 'a' : *dest++ = LITCHAR ; /*fall through*/
    case 'A' : *dest++ = SOH ;                         break;

    case 'o':                      /* superimpose \o'abcd' */

        get_quote("\\o") ;
        while((c = (* inp)(ifile)) != '\'' &&   maxch > 4)
        {
            maxch  -= 2 ;
            *dest++ = ZWIDTH;
            *dest++ = (c != Esc) ? c : (*inp)(ifile) ;
        }

        *dest++ = HMOVE  ;
        *dest++ = Hs_amt ;
        break;

        HORIZ++;

    case 'x':                     /*  \x<2-hex-digits>    */

        c = (*  inp)(ifile); /* c = MS digit         */
        i = (*  inp)(ifile); /* i = LS digit         */
        c = toupper(c);
        i = toupper(i);

        if( !ishex(c) || !ishex(i) )
            err("\\x must be followed by two hex digits\n");
        else
        {
            /*   \xNN takes up space. If you need to have
             *   a zero width escape sequence get to the
             *   output in right-adjusted text, use the
             *   .ou command or \fX mechanism.
             */

            *dest++ = (tohex(c) << 4)    |  tohex(i) ;
            HORIZ++;
        }
        break;

    case '{':

        *dest++ = '.';
        *dest++ = '{';
        HORIZ += 2;

        if( c = (*inp)(ifile) == Esc )
                c = (*inp)(ifile);

        goto exit;

    case '}':

        /* Setting Abort_process to nonzero forces
         * process() to terminate AFTER processing the
         * current line. It has the same effect as a .}
         * command at the beginning of  the next line.
         * All  text following the \{ on the line is
         * discarded.                   (})
         */

        Abort_process = 1;

        while(  (c = (*inp)(ifile)) != '\n'  &&  c != EOF )
                ;

        goto exit ;

    case 'e':                 /* \e printable version of the    */

        *dest++ = LITCHAR ; /*  current escape character        */
        *dest++ = Esc;
        HORIZ++;
        break;

    case 'f':               /* Change font \f(RBIOPx)      */

        if( maxch < 2 )
        {
            err("No room in input buffer\n");
            break;
        }

        switch( c = (*inp)(ifile) )
        {
        case BOLD:
        case ITALICS:
        case OVER:   *dest++ = CH_ATTRIB ; break;
        default:     *dest++ = CH_FONT   ; break;
        }

        *dest++ = c ;
        break;

    case 'r' :             /* up 1 line          */
    case 'u' :             /* up 1/2 line        */
    case 'd' :             /* down 1/2 line      */
    if( maxch > 2 )
        {
            *dest++ = VMOVE;
            *dest++ = (  c == 'r' ) ? - Vs_amt      :
                      (  c == 'u' ) ? -max( Vs_amt/2, 1) :
                      /* c == 'd' */    max( Vs_amt/2, 1) ;
        }
        break;

    case 'h' :  /* \h'N' \h'Nu' Horizontal motion */
    case 'v' :  /* \v'N' \v'Nu' vertical motion   */

        if( maxch < 2 )
                break;

        get_quote("\\v  or \\h") ;

        *dest++ = (c == 'v') ?  VMOVE : HMOVE ;
        j = c;
        i = gnum(inp, ifile, &c);

        if( c != 'u' )
            *dest++ = i * ((j=='v') ? Vs_amt : Hs_amt);
        else
        {
            *dest++ = i;
            get_quote("\\v  or \\h") ;
        }

        break;

    case 'l' :             /* horizontal line      \l'Nc' */
    case 'L' :             /* vertical line        \L'Nc' */

        /* Note that you can't  use an escape sequence for
         * the  line character (as in \l'10\x85'). You can
         * say:
         *              .ds li \\l'10\x85'
         *              \*(li
         * however.
         */

        get_quote("\\l'Nc' or \\L'Nc'");

        j = (c  == 'l');            /* j = 1 if horizontal */
        i = gnum(inp, ifile, &c) ; /* i = N in  \L'Nc'    */

        if( c == '\'' )
                linechar = j  ? '_'  :  '|' ;
        else
        {
                linechar = c;
                get_quote("\\l or \\L") ;
        }

        while(  --i >= 0  && --maxch > 4 )
        {
                if( !j )
                    *dest++ = ZWIDTH;

                *dest++ = linechar ;

                if( !j )
                {
                    *dest++ =   VMOVE  ;
                    *dest++ =   Vs_amt ;
                }
        }

        if( maxch < 0 )
            err("line drawn by \l or \L is too long\n");

        break;

    default :

        if( c == Hyphen_chr )                 /* \%           */
            *dest++ = SOFT_HYPHEN;
        else
            *dest++ = c ;                   /*  \<any char> */

        HORIZ++;
        break;
        break;
    }

newchar:
    c = ( *inp )( ifile );
exit:
    *target = dest;
    return( c );
}

/*-------------------------------------------------------
 *   Sgetc() is used to process a mode 2 process() call.
 */

sgetc(s)
UCHAR   **s;
{
        return **s ? *((*s)++) : EOF ;
}

/*------------------------------------------------------*/

chgfont( c )
{
        /* If New_font is non-zero a font-change
```

(continued on page 72)



The best new programs from one of the best-known publishers.

C++

ADVANTAGE C++

- New object-oriented language lets you develop large and complex programs with greater resilience, fewer bugs.
- Write reliable, reusable code that is easier to understand and maintain.
- Fully compatible with existing C programs and tools.
- All the benefits of C without its limitations.
- Available for Lattice and Microsoft C compilers.

Latest From LIFEBOAT

MULTI-TASKING

TimeSlicer

- Create multitasking and real-time aplications in C now!
- No need to interface with the OS. Tasks can be created, suspended or terminated at run-time.
- Optimize processor usage and transparency.
- Create more efficient and portable programs.
- Compatible with Lattice C, Microsoft C, ADVANTAGE C++ and object-oriented programming.
- Includes header files for C and assembly language and example programs with source code.

Latest From LIFEBOAT

LINKER

ADVANTAGE Link

- Fastest, most powerful PC-DOS overlay linker.
- First linker to take full advantage of EMS.
- Accepts Microsoft and Phoenix command files.
- Supports 53 commands; more than any other linker.
- Compatible with Microsoft CodeView and Pfix Plus.

Brand New LIFEBOAT

C COMPILER

ADVANTAGE C

- Newest technology based on AI and advanced design.
- Unmatched execution speed.
- Productive, easy-to-manage compilation environment.
- Advanced optimization routines; implements register variables.
- Over 250 library functions; full ANSI C library.
- Supports any size memory model (S,M,L).
- Microsoft C compatibility; recompiles most applications without source code changes.

Brand New LIFEBOAT

GRAPHICS

ADVANTAGE Graphics

- Complete C programming graphics library.
- Create bar and pie charts with one command.
- Automates easily.
- 10 popular font styles.
- No royalties or run-time fees.

Brand New LIFEBOAT

We make the best software even better.

After 10 years of publishing software, we know what's important to you. We're committed to full service that goes beyond just selling you the best software at competitive prices. Our expert staff can help you choose the right programs and provide full technical support. Count on Lifeboat for the complete solution to all your programming needs.

Call for the latest products from:

Blaise • Computer Innovations • CompuView • Creative Programming • Cytec • DMA • Essential • FairCom • Flexus • GSS • Gimpel • Greenleaf • Informix • Lattice • Microsoft • Media Cybernetics • Morgan Computing • Opt-Tech Data • Oregon Software • Periscope • Phar Lap • Phoenix • Rational Systems • Realia • Roundhill • SoftCraft • Software Bottling • Unipress • Vermont Creative • Wizard Systems • and more

Call **1-800-847-7078**

In NY **914-332-1875**

or your local Lifeboat Authorized Dealer.

The Full-Service Source for Programming Software.

LIFEBOAT

55 South Broadway
Tarrytown, NY 10591
Telex #510-601-7602

INTERNATIONAL SALES OFFICES

Australia/New Zealand: Charlton Distributors Phone: (64) (09) 766-361
Canada: Scantel Systems Phone: (416) 449-9252
England: Grey Matter, Ltd. Phone: (44) 364-53499
System Science, Ltd. Phone: (44) (01) 248-0962
France: Compusol Phone: (45) 30.07.37
Italy: Lifeboat Associates Italia Phone: (02) 464601
Japan: Lifeboat, Inc. Phone: (03) 293-4711
Spain: Micronet, S.A. Phone: (34) 1-262-3304
The Netherlands: SCOS Automation BV Phone: (31) 20-10 69 22
West Germany: MEMA Computer GmbH Phone: (69) 34-7226
Omnitex Phone: (76) 23-61820

The fastest, tightest code.

(Though the same can hardly be said of the name.)

We have to tell you, we had a hard time getting the name down this short.

Because Microsoft's new FORTRAN Compiler actually has a far longer list of features.

It uses the same optimizer and code generator technology that made our C Compiler the industry leader.

And we've added special loop optimizations that give you the smallest, fastest FORTRAN code a PC can handle.

> *"Now Microsoft's FORTRAN Optimizing Compiler generates such fast code that an IBM PC/XT approaches the speed of the VAX."*
>
> *Peter Osgood, MIT, Project Athena, Director of the Real Time Lab Project.*

This compiler has already passed the toughest test there is. It's been

---

Microsoft FORTRAN Optimizing Compiler Version 4.0.

- ♦ Full ANSI FORTRAN 77 Compiler with Fastest Executable Code for MS-DOS. NEW!
- ♦ Uses the Microsoft C optimizing technology, plus loop optimization. NEW!

Execution Speed (in Seconds)	Microsoft FORTRAN v. 4.0	Ryan-McFarland FORTRAN v. 2.11	IBM Professional FORTRAN v. 1.22
Sieve	7.97	9.33	38.51
Whetstone	53.82	58.67	79.04
Lookup	5.82	18.61	26.02

- ♦ Fully GSA certified for ANSI 77 compatibility with no errors at the highest level. NEW!
- ♦ Numerous IBM VS and DEC VAX extensions. NEW!
- ♦ Microsoft CodeView: Window-oriented source-level debugger. NEW!
  - —Debug using your original source code, the resulting disassembly or both intermingled.
  - —Watch and change the values of your local and COMMON variables as you debug.
  - —Set conditional breakpoints on variables, expressions or memory; trace and single step.
  - —Watch and change registers and flags as you execute.
  - —Easily debug graphics oriented programs since program output is kept separate from debugger output.

GSA-certified as Full ANSI FORTRAN 77, and 100% error-free.

"The Microsoft FORTRAN Optimizing Compiler let us port the 200,000 line Boeing Mathematical Library (BCSLIB) with virtually no changes. This ANSI FORTRAN 77 code was ported directly from Cray, CDC, DEC, IBM and other mainframes and workstations."

Ivor Philips, Boeing Computer Services, Program Manager Mathematical Software Libraries.

We've also included the same advanced intrinsic math functions found on VAX® and IBM® VS systems. Add

mpiler v. 4.0 with CodeView™

mprovements ke our new HUGE nemory model, and porting the iggest mainframe programs has ever been easier.

Among the many additions we've nade to our package is our exclusive odeView™ windowing debugger. It ts you trace through programs at ny level you want, from source ode to assembly language.

You can open windows, and watch both variables (local and COMMON) and CPU registers change. You can set conditional breakpoints using variables and expressions.

Debugging gets even easier with the compiler's advanced diagnostics. Detailed error messages are thoroughly explained and cross-referenced in our new manuals.

Documentation that has been completely revised and expanded with tons of examples.

If we're talking your language, use one of the numbers below for more details about the Microsoft® ANSI FORTRAN 77 Optimizing Compiler Version 4.0 with CodeView, and the name of your nearest dealer.

(Even if the call's toll-free, it may be a good idea to refer to it as "FORTRAN 4" for short.)

Microsoft® FORTRAN

The High Performance Software.

Call (800) 426-9400. In Washington State or Alaska, (206) 882-8088. In Canada, (416) 673-7638.

edium, Large and Huge Memory Model Libraries. NEW!
x models with NEAR, FAR and new HUGE pointers.
mmon blocks and arrays greater than 64K.
oose from three math libraries and generate in-line 8087/80287
tructions or floating point calls:
floating point emulator (utilizes 8087/80287 if installed)
8087/80287 coprocessor support
alternate math package—extra speed without an 8087/80287
nk your FORTRAN routines with Microsoft C (v. 4.0 or higher), Microsoft
scal (v. 3.3 or higher) or Microsoft Macro Assembler.
rgest number of 3rd party support libraries available.

- Provides more detailed diagnostic error messages (almost twice as many as competitors) and extensive documentation with non-ANSI 77 features highlighted. NEW!
- Proven reliability—tested with over 2.5 million lines of code compiled and executed.
- MS-DOS® network support with file / record locking and sharing.
- Microsoft Program Maintenance Utility rebuilds your applications after your source files have changed. NEW!
- Other utilities including faster overlay linker (links over 1Mbyte object code), library manager, EXE file compression utility, EXE file header utility, MS-DOS environment setting utility and setup utility.

Circle **no. 380** on reader service card.

Listing Eighteen

(Listing continued, text begins on page 96.)

```
        * request is appended to the front of the
        * next input line. Chgfont() is called
        * from the routine that processes the .ft
        * request [ ft() in nrprocs.c ] and also
        * when an environment is restored (by
        * pop_env() in nrmsc.c).
        */

        New_font = c;
}

/*------------------------------------------------------*/

int     getline( target, copymode, inp)
int     ( *inp )();
UCHAR   *target;
{
    /* Get an input line & put it into target. Get at most
     * MAXSTR characters. The input file, Ifile, is either
     * an input file or an input macro depending on the
     * state of the global Ismacro. Return 1 on success
     * or 0 on end of file. Lines ending with <Esc><newline>
     * are continued to the next line, otherwise newline
     * terminates the line. The newline character is not
     * put into the string. Tabs (^I) are expanded to a
     * sequence of spaces (\t is expanded into a ^I by
     * escape(). This ^I will be processed by in the text()
     * module. Trailing white space is stripped from the line.
     *
     * Copymode is just passed through to escape() (which
     * expands escape sequences).
     */

    register UCHAR  *rp ;
    register int    c   ;
    UCHAR           *p  ;

    if( Quit )          /* Quit is set by the .ex command  */
            return 0;   /* pretend we've seen end of file  */

    INLINES++ ;         /* Increment number of input lines */

    p = target ;
    c = (*inp)(Ifile);

    while( (p - target) < MAXSTR )
    {
            if( c == '\n' || c  == EOF )
                break;

            if( c != Esc )
            {
                *p++ = c   ;
                c = (   *inp )( Ifile ) ;
            }
            else
                c = escape(target, &p, copymode, inp,   Ifile,
                                         MAXSTR - (p-target));

    }

    *p = 0;
    if( Tabs_enabled )
            dotab( target );    /* Expand tabs and leaders */

    if( New_font && !ISCMD(*target) )
    {
            /* This is a kludge but it's the most convenient
             * way to get a font change into the input stream
             * at the correct place. We can't just change
             * fonts when the .ft is executed because we
             * might be filling and .ft doesn't cause a break
             */

            memcpy( target+2, target, (p-target) + 1 );
            switch(New_font)
            {
            case PREVIOUS:
            case BOLD:
            case ITALICS:
            case OVER:          target[0] = CH_ATTRIB;
                                break;

            default:            target[0] = CH_FONT;
                                break;

            }

            target[1] = New_font ;
            New_font  = 0;
    }

    return( !(c == EOF && p == target) );
}

/*------------------------------------------------------*/

process( nifile, nifilename, mode, nmacv )                    */
FILE    *nifile ;          /* Input file descriptor           */
char    *nifilename ;      /* Name of input file              */
int     mode;              /* processing mode (see below)     */
char    **nmacv ;          /* Macro arguments
{
    /* This routine actually does the processing of a file
     * or a macro. It is a 2nd order recursive routine. That
     * is process() is called recursively every time the
     * input is changed (by a .so command, a macro expansion,
     * etc.). Macv is an argv-like array of arguments to
     * macros. Ifile is a pointer to either a FILE or to a
```


(continued on page 74)

UNIFY DBMS/DOS. The UNIX World Leader Brings A New Dimension To DOS Application Development.

What happens as the DOS world expands? As a new generation of hardware takes over? As networking becomes more important? The potential is enormous. But until now, the tools to achieve it have been limited.

Now a leader from another world unleashes that potential: UNIFY® DBMS. The leading relational DBMS in the UNIX™ world. And now, the most advanced set of application development tools in the DOS world.

With UNIFY DBMS, DOS developers have new power to build more sophisticated applications than ever before possible.

The power to write high performance "C" programs that will access the data base, using Unify's Direct Host Language Interface.

The power of an industry standard query language—SQL.

The power of unmatched speed in production applications. Only UNIFY DBMS is specifically engineered for transaction throughput. With unique performance features like PathFinder™ Architecture multiple access methods, for the fastest possible data base access.

The power of comprehensive program development and screen management tools. Plus a state-of-the-art fourth generation report-writer.

What's more, with UNIFY DBMS, the potential of networked applications becomes a reality. Unlike DBMS systems which were originally single-user (and which have a long stretch to accommodate more users), UNIFY DBMS is a *proven* multi-user system.

And because UNIFY DBMS/DOS is the best of two worlds, it offers you the most powerful benefit of all: DBMS applications that can grow as your needs grow. From single user DOS. To networked DOS. To multi-user UNIX. All without changing your applications.

Call the Unify Information Hotline for our free booklet: The New DOS World. (503) 635-7777

UNIFY
CORPORATION
4000 Kruse Way Place
Lake Oswego, OR 97034

 Circle no. **332** on reader service card.

Listing Eighteen

(Listing continued, text begins on page 96.)

```
 * macro descriptor returned by mopen() (in nrmac.c)
 * according to the state of mode. The following modes
 * are recognized
 *
 *   0    Input is from a file or stream and Ifile is
 *        a FILE pointer.
 *   1    Input is from a macro and Ifile is a macro
 *        pointer returned from mopen().
 *   2    Input is from a string and Ifile is a pointer
 *        to that string. Note that mode two commands
 *        are processed with the current file (ie.
 *        ifile, nifilename and nmacv are ignored).
 *
 * Process returns immediatly if command() returns true.
 *
 * This routine is extremely recursive. Be careful with
 * static variables (ie. don't use them).
 */

    UCHAR   line [MAXSTR]       ;
    UCHAR   *oiname, **omacv    ;
    int     oinlines, oismacro  ;
    FILE    *oifile  ;
    int     oinhibit ;
    int     mgetc(), fgetc();

#ifdef DEBUG
    printf("Mode %d process call:  ", mode );
    printf("file <0x%x> named <%s>, macv @ 0x%x\n",
                            nifile, nifilename,     nmacv );

    printf("%s processing started (from %s, line %d)\n",
            nifilename, Ifilename, INLINES );
#endif

    oinhibit   = Inhibit    ;

    oinlines   = INLINES    ;  /* Save program state on   */
    oifile     = Ifile      ;  /* the stack               */
    oiname     = Ifilename  ;
    oismacro   = Ismacro    ;
    omacv      = Macv       ;

    INLINES   = 0           ; /* Create new program state */
    Ifile     = nifile      ;
    Ifilename = nifilename  ;
    Ismacro   = mode        ;
    Macv      = nmacv       ;

    if( mode == 2 )
    {
        Ifile = (FILE *) &nifile;  /* Ifile is  a string ptr */
        getline( line,  0, sgetc );

        if( Verbose )
            printf( "\n%s:<%s>\n", Ifilename, line );

        Inhibit   = oinhibit ;
        INLINES   = oinlines ;
        Ifilename = oiname   ;
        Ifile      = oifile   ;
        Ismacro   = oismacro ;
        Macv       = omacv    ;

        if( ISCMD( *line )  )
            command( line );
        else
            text( line  );
     }
     else
     {
         while(  getline(line, 0, Ismacro ? mgetc : fgetc) )
         {
             if( Verbose )
                 printf( "\n%s:<%s>\n", Ifilename, line );

             if( *line == FF )
                 command( ".bp" );

             else if( !  ISCMD(*line)  )
                 text(   line );

             else if(  command(line)  )
                 break;

             if( Abort_process )
             {                                   /*{*/
                 Abort_process   = 0;
                 if( command(".}") )
                     break;
             }
         }

         Inhibit   = oinhibit ;
         INLINES   = oinlines ;
         Ifilename = oiname   ;
         Ifile      = oifile   ;
         Ismacro   = oismacro ;
         Macv       = omacv    ;
     }

#ifdef DEBUG
   printf("%s: processing done (returning to %s, line %d)\n",
                    nifilename, Ifilename, INLINES );
#endif

    return 0;
}
```

End Listings

Turn the page for our latest advertised price list.

apl language

APL*PLUS/PC by STSC	595	429
APL*PLUS/PC Spreadsheet Mgr by STSC	195	139
APL*PLUS/PC Tools Vol 1 by STSC	295	199
APL*PLUS/PC Tools Vol 2 by STSC	85	59
Financial/Statistical Library by STSC	275	195
Pocket APL by STSC	95	69
STATGRAPHICS by STSC	795	579

artificial intelligence

1st-CLASS by Programs in Motion	495	399
APT from Solution Systems	65	CALL
Arity Combination Package	1225	1119
Expert System Development Pkg	295	259
File Interchange Toolkit	50	45
PROLOG Compiler & Interpreter	795	699
Screen Design Toolkit	50	45
SQL Development Package	295	259
Arity PROLOG Interpreter	350	309
Arity Standard Prolog	95	79
AutoIntelligence by IntelligenceWare	990	CALL
ExpertEDGE Advanced by Human Edge	2500	CALL
ExpertEDGE Professional by Human Edge	5000	CALL
Experteach II by IntelligenceWare	475	359
EXSYS Development Software by EXSYS	395	319
EXSYS Runtime System New	600	479
GCLISP Golden Common LISP by Gold Hill	495	CALL
GCLISP 286 Developer by Gold Hill	1190	CALL
Insight 1 by Level Five Research	95	75
Insight 2+ by Level Five Research	485	379
Intelligence/Compiler IntelligenceWare	990	749
Logic-Line Series 1 by Thunderstone	90	85
Logic-Line Series 2 by Thunderstone	125	115
Logic-Line Series 3 by Thunderstone	150	139
LPA microPROLOG All Varieties	CALL	CALL
Microsoft LISP Common LISP	250	163
MPROLOG Language Primer LOGICWARE New	50	45
MPROLOG P500 by LOGICWARE New	495	395
MPROLOG P550 by LOGICWARE New	220	175
PC Scheme by Texas Instruments	95	84
Personal Consultant Easy by TI	495	435
Personal Consultant Plus by TI	2950	2589
Personal Consultant Runtime	95	85
QNIAL by NIAL Systems	375	349
TransLISP from Solution Systems	95	CALL
TransLISP PLUS from Solution Systems New	195	CALL
Turbo PROLOG by Borland Intl New Version	100	65

assembly language

386 ASM/LINK Cross Asm by Phar Lap	495	389
8088 Assembler w/Z-80 Trans by 2500 AD	100	89
ASMLIB Function Library by BC Assoc	149	129
asmTREE B-Tree Dev System by BC Assoc	395	339
Cross Assemblers Various by 2500 AD	CALL	CALL
Microsoft Macro Assembler	150	95
Norton Utilities by Peter Norton	100	59
Turbo EDITASM by Speedware	99	84
Uniware Cross Assemblers Various by SDS	295	249
Visible Computer: 8088 Software Masters	80	65

basic language

BetterBASIC by Summit Software Clearance	200	119
EXIM Services Toolkit by EXIM	CALL	CALL
Finally by Komputerwerks	99	85
Inside Track from Micro Help	65	51
MACH 2 by Micro Help	75	59
Microsoft QuickBASIC	99	65
87 QB Pak by Hauppauge New	69	59
Peeks 'n Pokes from MicroHelp	45	37
Professional BASIC by Morgan	99	75
8087 Math Support	50	42
Stay-Res by MicroHelp	95	74
True Basic New Version	150	99
True Basic w/Run-time	245	179
BASICA Converter	50	45
Run-time Module	150	99
Various Other Utilities	50	45
Turbo BASIC by Borland Intl New	100	69

blaise products

ASYNCH MANAGER Specify C or Pascal	175	124
C TOOLS PLUS	175	124
EXEC Program Chainer	95	75
LIGHT TOOLS for Datalight C New	100	89
PASCAL TOOLS	125	99
PASCAL TOOLS 2	100	79
PASCAL TOOLS & PASCAL TOOLS 2	175	124
RUNOFF Text Formatter	50	45
TURBO ASYNCH PLUS	100	79
TURBO POWER TOOLS PLUS	100	79
VIEW MANAGER Specify C or Pascal	275	189

borland products

EUREKA Equation Solver New	100	69
REFLEX & REFLEX Workshop	200	129
REFLEX Data Base System	150	89
REFLEX Workshop	70	45
Turbo BASIC New	100	69
Turbo DATABASE TOOLBOX	70	47
Turbo EDITOR TOOLBOX	70	47
Turbo GAMEWORKS TOOLBOX	70	47
Turbo GRAPHIX TOOLBOX	70	47
Turbo LIGHTNING	100	64
Turbo Numerical Methods Library New	100	69
Turbo PASCAL and TUTOR New	125	85
Turbo PASCAL with 8087 and BCD	100	64
Turbo TUTOR	40	28
Turbo Prolog Compiler New Version	100	64
Turbo Prolog Toolbox New	100	64
Word Wizard	70	47
Word Wizard and Turbo Lightning	150	94

c++

C++ by Guidelines w/kernel 1.1	195	179
PforCe++ Library for C++ by Phoenix New	395	229

c compilers

68000/10/20 Cross Compiler by SDS	595	CALL
C86PLUS by Computer Innovations New	497	CALL
Datalight C Compiler Small Model	60	49
Datalight Developer Kit	99	79
Datalight Optimum-C New	139	119
DeSmet C w/Debugger	159	138
DeSmet C w/Debugger & Large Case	209	184
Eco-C Development System by Ecosoft	125	83
Lattice C Compiler from Lattice	500	275
Mark Williams Let's C Combo Pack New	125	99
Let's C Compiler	75	57
csd Source Level Debugger	75	57
Mark Williams MWC-86	495	289
Microsoft C with CodeView	450	275
Wizard C Combo by Wizard Systems	750	529
Wizard C Compiler	450	299
ROM Development Pkg	350	259

c interpreters

C-terp by Gimpel, Specify compiler	300	235
C Trainer with Book by Catalytix	122	87
Instant C by Rational Systems	500	379
Introducing C by Computer Innovations	125	CALL
Run/C from Lifeboat	150	88
Run/C Professional from Lifeboat	250	159

c utilities

C Essentials by Essential Software	100	75
C-ISAM by Informix	225	195
C to dBase by Computer Innovations	150	CALL
c-tree & r-tree Combo by FairCom	650	529
c-tree ISAM File Manager	395	329
r-tree Report Generator	295	249
C Utility Library by Essential	185	135
C Windows by Syscom	100	85
C Wings by Syscom	50	43
CI ROMPac by Computer Innovations	195	CALL
dbQUERY All Varieties by Raima	CALL	CALL
dbVISTA Single-User DBMS by Raima	195	155
with Source Code	495	425
dbVISTA Multi-User DBMS by Raima	495	425
with Source Code	990	845
dBx dBase/C Translator by Desktop AI	350	314
with Library Source Code	550	493
Entelekon Combo Package	200	169
C Function Library	130	109
C Windows	130	109
Superfonts for C	50	43
Essential Comm Library w/Debugger	250	195
Breakout Debugger Any language	125	99
Essential Comm Library	185	135
Essential Graphics by Essential Software	250	195
Flash-up Windows by Software Bottling	90	79
GraphiC Mono v2.2 by Sci Endeavors	280	209
GraphiC Color v3.0 by Sci Endeavors	350	284
GRAFLIB by The Librarian	175	CALL
Greenleaf Comm Library by Greenleaf	185	127
Greenleaf Data Windows by Greenleaf	225	157
with Source Code	450	295
Greenleaf Functions by Greenleaf	185	127
HALO by Media Cybernetics	300	209
HALO Development Pkg for Microsoft New	595	395
The HAMMER by OES Systems	195	139
HELP/Control by MDS	125	109
MetaWINDOWS No Royalties	185	115
MetaFONTS	80	58
MetaWINDOWS/Plus by Metagraphics	235	189
MetaFONTS/Plus	235	189
PANEL by Roundhill Computer Systems	295	215
PC Lint by Gimpel Software	139	99
PLOTHI by The Librarian	175	CALL
PLOTHP by The Librarian	175	CALL
Scientific Subroutine Lib by Peerless	175	134
screenplay by Flexus	175	129
Vector87 by Vectorplex Data Systems	150	135
Vitamin C by Creative Programming	225	CALL
VC Screen Forms Designer	100	82
Zview by Data Management Consultants	245	189

cobol language

Micro Focus COBOL Workbench	4000	CALL
Micro Focus Level II COBOL	1500	CALL
COGRAPHICS	250	CALL
COMATH	200	CALL
FORMS-2	300	CALL
Level II Animator	900	CALL
Level II SOURCEWRITER	2000	CALL
Micro Focus Level II COBOL for Novell	2000	CALL
Micro Focus Professional COBOL	3000	CALL
Multi-user Runtime for PC Network	500	CALL
Microsoft COBOL See Microsoft Section	CALL	CALL
Realia COBOL	995	785
Realia CICS New	995	785
RM/COBOL by Ryan-McFarland	950	639
RM/COBOL 85 by Ryan-McFarland	1250	895
screenplay by Flexus	175	129

debuggers & profilers

386 DEBUG Cross Debugger by Phar Lap	195	129
Advanced Trace-86 by Morgan Computing	175	125
CI Probe by Computer Innovations	225	CALL
Codesifter Profiler by David Smith	119	94
Codesmith-86 by Visual Age	145	99
DSD86 by Soft Advances	70	61
DSD87 by Soft Advances	100	79
MiniProbe by Atron New	395	CALL
Periscope I by The Periscope Company	345	CALL
Periscope II w/NMI Breakout Switch	175	CALL
Periscope II-X Software only	145	CALL
The PROFILER with Source Code by DWB	125	89
The WATCHER Profiler by Stony Brook	60	51

forth language

CFORTH Native Code Compiler by LMI	300	229
Forth/83 Metacompiler Specify Target	750	599
PC/Forth by Laboratory Microsystems	150	109
PC/Forth+ by Laboratory Microsystems	250	199
Advanced Color Graphics Support	100	74
Enhanced Graphics Support	200	148
Intel 8087 Support	100	74
Interactive Symbolic Debugger	100	74
Native Code Optimizer	200	148
Software Floating Point	100	74
UR/Forth and support utilities by LMI	CALL	CALL

fortran language

50 MORE: FORTRAN by Peerless Engr	125	95
ACS Time Series Alpha Computer Service	495	399
Btrieve ISAM File Mgr by SoftCraft	245	194
Essential Graphics by Essential Software	250	195
For-Winds Alpha Computer Service	90	69
Forlib-Plus Alpha Computer Service	70	49
FORTLIB by Sutrasoft	95	CALL
FORTRAN Addenda by Impulse Engr	95	85
FORTRAN Addendum by Impulse Engr	165	139
GRAFLIB by Sutrasoft	175	CALL
HALO by Media Cybernetics	300	209
I/O PRO by MEF Environmental	149	129
Microcompatibles Combo Package	240	219
Grafmatic	135	119
Plotmatic	135	119
Microsoft FORTRAN w/CodeView - New Version	450	CALL
No Limit by MEF Environmental	129	115
PANEL Screen Designer by Roundhill	295	215
PLOTHI by Sutrasoft	175	CALL
PLOTHP by Sutrasoft	175	CALL
RM/FORTRAN Ryan-McFarland	595	CALL
Scientific Subroutine Lib by Peerless	175	134
Statistician Alpha Computer Service	295	245
Strings & Things Alpha Computer Service	70	51
Vector87 by Vectorplex Data Systems	150	135

lattice products

Lattice C Compiler from Lattice	500	275
with Library Source Code	900	495
C Cross Reference Generator	50	37
with Source Code	200	145
C-Food Smorgasbord Function Library	150	95
with Source Code	300	184
C-Sprite Source Level Debugger	175	129
Curses Screen Manager	125	89
with Source Code	250	178
dBC dBase File Manager for C	250	178
with Source Code	500	356
LMK Make Facility	195	139
RPG II Compiler No Royalties	750	626
RPG II Combo with SEU & Sort/Merge	1100	939
RPG II Screen Design Aid Utility New	350	309
SecretDisk File Encryption Utility	120	89
SideTalk Resident Communications	120	89
SSP/PC Scientific Library New	350	269
Text Management Utilities	120	89
TopView Toolbasket Function Library	250	178
with Source Code	500	356

logitech products

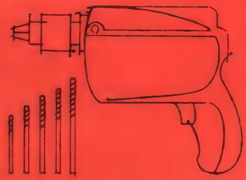
LOGIMOUSE C7 Specify Connector Type	99	83
with PLUS Pkg	119	98
with PLUS & PC Paintbrush	169	134
with PLUS & CAD Software	189	153
with PLUS & Reflex	199	162
with PLUS & CAD & Paint	219	179
with PLUS & CAD & Paint & Reflex	299	245
LOGIMOUSE BUS with PLUS Pkg New	139	115
with PLUS & PC Paintbrush New	189	149
with PLUS & CAD Software New	209	175
with PLUS & Reflex New	219	179
with PLUS & CAD & Paint New	239	195
with PLUS & CAD & Paint & Reflex New	319	259
MODULA-2/86 Holiday Package	199	159
MODULA-2/86 Compiler	89	62
MODULA-2/86 with 8087 Support	129	98

Listing One *(Listing continued, text begins on page 110.)*

```
*-----------------------------------------------------------------------*
*  paste -- a program to attach to the lines of a file the correspond-  *
*        ing lines of another file, with an optional string between     *
*        them.                                                          *
*                                                                       *
*  Written January, 1984 by John M. Gamble                              *
*  Updated for UNIX April, 1986                                         *
*                                                                       *
*  usage:                                                               *
*                                                                       *
*  paste [-paste] [-b <string>] [-<n>] [file1] [file2]                  *
*                                                                       *
*  options:                                                             *
*                                                                       *
*  -p     <file1> does not exist (<string> is prepended to each         *
*         line.)                                                        *
*                                                                       *
*  -a     <file2> does not exist (<string> is appended to each          *
*         line.)                                                        *
*                                                                       *
*  -s     Do not print <string> with lines from only one file.          *
*                                                                       *
*  -t     An option to resolve the ambiguous command                    *
*         "paste <file>".       The -t flag forces <file> to trail      *
*         standard input.  I.e.,                                        *
*                                                                       *
*         "paste <file>"                                                *
*                 is equivalent to "paste <file> <stdin>"               *
*                                                                       *
*         "paste -t <file>"                                             *
*                 is equivalent to "paste <stdin> <file>".              *
*                                                                       *
*  -e     Do not print <string> if both input lines are empty           *
*         (i.e., that consist of no characters but '\n'.)               *
*                                                                       *
*  -b <string> A string of characters to be inserted between the lines *
*              of <file1> and <file2>.  The string may contain all      *
*              the standard escape codes with the exception of '\0'.    *
*              The string may also indicate blanks with the escape      *
*              sequence '\s'.                                           *
*                                                                       *
*  -<n>   Print <n> lines of <file1> before appending lines of          *
*              <file2>.  If <n> is negative (e.g., "paste --3") then    *
*                <n> lines of <file2> will be printed first.            *
*-----------------------------------------------------------------------*/

#include  <stdio.h>
#include  <ctype.h>

/* On systems such as UNIX, if a string with blanks in it is
 * surrounded by quote marks, it is considered to be one string.
 * On other systems, the blank ends the string and the quote
 * marks are passed along with the other characters.  So, while
 * on UNIX, the command
 *
 *          'paste -b "; do " list1 list2'
 *
 * would set
 *          argv[2] to "; do ",
 *          argv[3] to list1,
 *          argv[3] to list2.
 *
 * a system like MSDOS would set
 *          argv[2] to "\";",
 *          argv[3] to "do",
 *          argv[4] to "\"",
 *          argv[5] to list1,
 *          argv[6] to list2.
 *
 * This is easily taken care of, but it does mean that conditional
 * compilation is required by setting the switch below to either
 * zero or one, depending on your particular operating system.
 */
#define BLANK_ENDS_STR                          0

#define STRINGLEN                       128
#define TRUE                                    1
#define FALSE                                   0

#define isoctal(x)                      ((x) >= '0' && (x) < '8')

typedef unsigned int                    Boolean;

char        bstring[STRINGLEN] = {'\0'};

char        *nullstr = "";
char        *strf = "%s";
char        *program_name = "paste";
char        *error_msg[] =
        {
/*0*/       "usage: %s [-aptse] [-b \"string\"] [-<n>] [file1] [file2] %s\n",
/*1*/       "%s: unknown flag %s\n",
/*2*/       "%s: at least one file must exist%s\n",
/*3*/       "%s: -t flag is only valid with one file on the command line%s\n",
/*4*/       "%s: both files can't be standard input%s\n",
/*5*/       "%s: contradictory options%s\n",
/*6*/       "%s: can't open %s\n",
/*7*/       "%s: -a or -p flags are invalid with two files%s\n",
/*8*/       "%s: too many files%s\n",
/*9*/       "%s: string argument lacks closing \' or \"%s\n",
/*10*/      "%s: null character in string argument%s\n",
/*11*/      "%s: string argument too long%s\n"
            };

main(argc, argv)
int                 argc;
char                **argv;
{
```

```c
	FILE		*fp1, *fp2, *fopen();
	Booleanprepend = FALSE, append = FALSE,  trailing = FALSE;
	Booleanprintempty = TRUE, printsingle = TRUE;
	int		slip = 0;
	char		*subarg;

	if (argc == 1)
	   exit_error(0, nullstr);

	/* Get the flags.
	 */
	while (--argc > 0 && **++argv == '-')
	{
	   switch (*(*argv + 1))
	   {
	      case '\0':		/* Because default: won't catch this.*/
	         exit_error(1, *argv);
	         break;

	      case 'b':
	         if (argc == 1)
	            exit_error(0, nullstr);

	         argc--;
	         argv++;
#if BLANK_ENDS_STR
	         strget(&argc, &argv, bstring);
#else
	         strload(*argv, bstring);
#endif
	         break;

	      case '-':
	      case '0':
	      case '1':
	      case '2':
	      case '3':
	      case '4':
	      case '5':
	      case '6':
	      case '7':
	      case '8':
	      case '9':
	         slip = atoi(*argv + 1);
	         break;

	      default:
	         subarg = *argv;
	         while (*++subarg)
	         {
	            switch (*subarg)
	            {
	               case 'a':
	                  append = TRUE;
	                  break;

	               case 'e':
	                  printempty = FALSE;
	                  break;

	               case 'p':
	                  prepend = TRUE;
	                  break;

	               case 's':
	                  printsingle = FALSE;
	                  break;

	               case 't':
	                  trailing = TRUE;
	                  break;

	               default:
	                  exit_error(1, *argv);
	                  break;
	            }
	         break;
	      }
	   }
	}
	if (prepend && append)		/* Contradictory options.*/
	   exit_error(2, nullstr);

	switch (argc)	/* The number of file names on the command line.*/
	{
	   case 0:
	      if (trailing)
	         exit_error(3, nullstr);

	      if (!(prepend || append))		/* Both files can't be stdin.*/
	         exit_error(4, nullstr);

	      if (append)
	         attachf(stdin, NULL, slip, printsingle, printempty);
	      else
	         attachf(NULL, stdin, slip, printsingle, printempty);
	      break;

	   case 1:

	      /* Contradictory options?
	       */
	      if (trailing && (prepend || append))
	         exit_error(5, nullstr);

	      if ((fp1 = fopen(*argv, "r")) == NULL)
	         exit_error(6, *argv);
```

(continued on page 82)



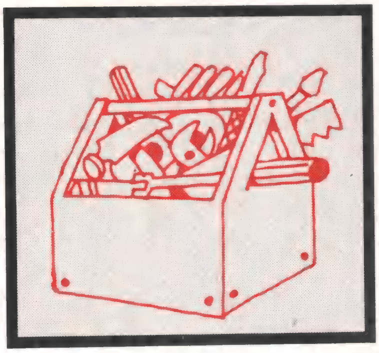
POWER TOOLS for SYSTEM BUILDERS™

800-543-6277

Ask for Operator 2053
California: 800-368-7600

TSF is owned and operated by programmers, so we understand your needs. We believe *and practice* integrity in all business dealings. We advertise real prices and offer our best price to all customers. We provide prompt delivery of current product versions.

We accept checks, Visa, MasterCard, American Express and COD. We charge your card only when we ship and do **not** add a surcharge. We provide free UPS delivery on orders over $100 ($3 delivery charge for small orders). Our COD fee is $4. We allow return privileges on most products. ***Give us a try!***



The Software Family
649 Mission Street
San Francisco, CA 94105

See us at the Computer Faire, Moscone Center, San Francisco, March 26 through 29, 1987!

Winter Specials

Basic Language

Microsoft Quick Basic (List $99)	$65
MicroHelp Inside Track ($65)	$55
MicroHelp Mach 2 (List $75)	$60
MicroHelp Stay-Res (List $95)	$75
Sterling Castle Tools (List $99)	$80

C Language

Datalight C (List $99)	$75
Microsoft C w/Codeview ($450)	$270
Guidelines C++ (List $195)	$175
Lattice C (List $500)	$270
Mark Williams C (List $500)	$290
Rational Instant C	$375
Gimple C-Terp (List $300)	$224
Run/C Professional (List $250)	$160
Run/C Interpreter (List $120)	$85
Gimpel PC Lint (List $139)	$103
Softcraft Btrieve (List $245)	$190
Lattice dBase III Intfc ($250)	$175
Blaise View Manager (List $275)	$197
Vermont Window for Data ($295)	$259
Lattice Curses (List $125)	$90
Essentials C Essentials ($100)	$80
Essentails Graphics (List $250)	$200
Blaise C Tools+ (List $175)	$125
Essentials Utility (List $185)	$130
Greenleaf Functions (List $185)	$130
Greenleaf Comm (List $185)	$130
Blaise Async Manager ($175)	$125

... and many more

Pascal Language

Microsoft Pascal (List $300)	$180
Blaise View Manger (List $275)	$197
Blaise Async Manager ($175)	$125
Blaise Pascal Tools+ (List $175)	$125

Turbo Pascal

Turbo Pascal BCD & 8087 ($100)	$65
Software Channels Alice ($95)	$70
Kydor Symbolic Debugger ($49)	$40
Blaise Turbo Async ($100)	$80
Blaise Power Tools ($100)	$80
Borland Turbo Editor (List $70)	$48

Other Tools

Custom Software PC/VI ($149)	$119
MKS Toolkit ($List $139)	$115
Periscope I (List $295)	$235
Periscope II (List $145)	$108
Phoenix Plink+ (List $495)	$315
Phoenix Pfinish (List $395)	$230
Phoenix Pfix86+ (List $395)	$225
Polytron Make (LIst $99)	$80
Quilt SRMS + Qmake (List $199)	$165
Seidl SVM + SMK (List $379)	$330
Dan Bricklin's Demo (List $75)	$68
Opt-Tech Sort (List $149)	$126
Borland Turbo Prolog (List $99)	$65
Microsoft MASM (List $150)	$98
Microsoft Fortran (List $350)	$200

... call for price list

Basic: ***Mach2*** from ***MicroHelp*** expands Basic's horizons with window management, input editing, array sorting and print formatting. It also provides MSDOS/BIOS function calls and allows you to break the 64K data barrier. Written in assembler for high performance. For Bascom, Quick Basic, BASICA and GWBASIC. (List $75) Only $60 from TSF. 30 day money back guarantee!

C: ***Gimpel's PC-Lint*** evaluates your source code to locate insidious C bugs like "=" instead of "==" and mismatched function parameters. Evaluates multiple source files to validate use of all public functions and variables. Compatible with all popular compilers. 30 day trial period (15% re-stock fee). (List $139) Only $103 from **TSF.**

All Languages: ***Periscope II*** speeds debugging with multiple windows, symbolic memory address-ing, and sophisticated trace and breakpoints. Includes an NMI break switch so you can interrupt programs at any time without pre-setting break points -- ends guesswork for hard-to-locate crashes. (List $145) Only $108 from **TSF.** 30 day money back guarantee!

Turbo: ***Kydor's Symbolic Debugger*** provides symbolic debugging for Turbo Pascal programs. Includes trace, breakpoint, memory display and memory modification. Runs from within Turbo's environment, so you don't loose any Turbo features. Written in assembler for speedy operation and low memory overhead. (List $49) Only $40 from **TSF.**

Communications: ***RamNet*** from ***System Design*** provides back-ground file transfers and e-mail for your PC. Incoming and outgoing calls are processed automatically while you con-tinue with your normal work. Outgoing calls can be scheduled to take advan-tage of late night phone rates and to concentrate messages in a multi-node ***RamNet*** network. (List $149) Only $129 from **TSF.** A two node starter kit is only $248. 30 day money back guarantee!

QUILT COMPUTING · MICROSOFT · BORLAND INTERNATIONAL · Lattice · Phoenix · BLAISE COMPUTING

Circle **no. 230** on reader service card.

Listing One *(Listing continued, text begins on page 110.)*

```
        if (append)
            attachf(fpl, NULL, slip, printsingle, printempty);

        else if (prepend)
            attachf(NULL, fpl, slip, printsingle, printempty);

        else if (trailing)
            attachf(stdin, fpl, slip, printsingle, printempty);

        else
            attachf(fpl, stdin, slip, printsingle, printempty);

        fclose(fpl);
        break;

    case 2:
        if (trailing)
            exit_error(3, nullstr);

        if (prepend || append)
            exit_error(7, nullstr);

        if ((fpl = fopen(*argv, "r")) == NULL)
            exit_error(6, *argv);

        if ((fp2 = fopen(*++argv, "r")) == NULL)
            exit_error(6, *argv);

        attachf(fpl, fp2, slip, printsingle, printempty);
        fclose(fpl);
        fclose(fp2);
        break;

    default:
        exit_error(8, nullstr);
        break;
    }

    exit(0);                                        /* End of main.*/
}

#if BLANK_ENDS_STR
/*--------------------------------------------------------------------*
 * strget -- retrieve the <string> argument from the command line.    *
 *      If the string contains blanks, C assumes this is the end of the *
 *      string, and places a \0 at its end.  Since WE know that it's  *
 *      just a blank, we put one in, update the position of argv, and *
 *      decrement argc.  Escape sequences are treated just as defined *
 *      in C (except \0, which is an error).  One extra escape sequence *
 *      ('\s') exists to handle multiple blanks on a line, for even if *
 *      the string is enclosed in quotes the extra blanks will not be *
 *      passed from the command line.                                 *
 *--------------------------------------------------------------------*/
strget(pargc, pargv, bstr)
int                 *pargc;
char                ***pargv;
char                *bstr;
{
    register int        j;
    char                *subarg;
    Boolean st_quote = FALSE;
    Boolean st_apost = FALSE;

    subarg = **pargv;

    /* If the string is begun with a quote or an apostrophe, remember
     * so that we know when to end the string.
     */
    if ((st_quote = (*subarg == '"')) || (st_apost = (*subarg == '\'')))
        subarg++;

    for (j = 0; j < STRINGLEN; bstr++, subarg++, j++)
    {
        /* A `"` or `'` encountered could mean the end of a string -
         * check against st_quote or st_apost.
         */
        if ((st_quote && *subarg == '"') ||
            (st_apost && *subarg == '\''))
            break;
                                            /* Blank encountered in string.*/
        else if (*subarg == '\0')
        {

            /* If we began with a quote, we are not finished.
             */
            if (st_quote || st_apost)
            {
                /* If nothing is left on the command line,
                 * a quote mark is missing.
                 */
                if (--(*pargc) == 0)
                    exit_error(9, nullstr);

                /* Put the blank in, and point subarg
                 * to the next argv string.
                 */
                *bstr = ' ';
                subarg = *(++(*pargv)) - 1;
            }

            /* Otherwise we didn't start with a quote mark - end.
             */
            else
                break;
        }
        else if (*subarg == '\\')                   /* Escape sequences.*/
            switch(*++subarg)
```

(continued on page 84)

Listing One *(Listing continued, text begins on page 110.)*

```
                {
                    /* Nothing after the '\' - let the 'blank'
                     * section handle it.
                     */
                    case '\0':
                        bstr--;
                        subarg--;
                        j--;
                        break;

                    case '"':
                        *bstr = '"';
                        break;

                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
                    case '7':
                        *bstr = (char) bit_pattern(&subarg);
                        break;

                    case '\\':
                        *bstr = '\\';
                        break;

                    case 'b':
                        *bstr = '\b';
                        break;

                    case 'f':
                        *bstr = '\f';
                        break;

                    case 'n':
                        *bstr = '\n';
                        break;

                    case 'r':
                        *bstr = '\r';
                        break;

                    case 't':
                        *bstr = '\t';
                        break;

                    case 's':
                        *bstr = ' ';
                        break;

                    default:
                        *bstr = *subarg;
                        break;
                }

            else
                *bstr = *subarg;        /* No special character handling.*/
        }

        if (j == STRINGLEN)
            exit_error(11, nullstr);

        *bstr = '\0';                               /* End of strget.*/
}
#else

/*-------------------------------------------------------------------*
 * strload -- retrieve the <string> argument from the command line.  *
 *       Escape sequences are treated just as defined in C (except \0, *
 *       which is an error).  One extra escape sequence ('\s') exists in *
 *       order to handle multiple blanks on a line without bothering to *
 *       enclose the string in quotes.                                 *
 *-------------------------------------------------------------------*/
strload(subarg, bstr)
char                    *subarg;
char                    *bstr;
{
    extern char         *nullstr;
    register int        j;


    for (j = 0; *subarg && j < STRINGLEN; bstr++, subarg++, j++)

        if (*subarg == '\\')        /* Escape sequences.*/
            switch(*++subarg)
            {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                    *bstr = (char) bit_pattern(&subarg);
                    break;

                case '\\':
                    *bstr = '\\';
                    break;
                    case 'b':
                        *bstr = '\b';
                        break;
```

```
            case 'f':
               *bstr = '\f';
               break;

            case 'n':
               *bstr = '\n';
               break;

            case 'r':
               *bstr = '\r';
               break;

            case 't':
               *bstr = '\t';
               break;

            case 's':
               *bstr = ' ';
               break;

            default:
               *bstr = *subarg;
               break;
            }
      else
         *bstr = *subarg;        /* No special character handling.*/

   if (j == STRINGLEN)
      exit_error(11, nullstr);

   *bstr = '\0';
}
#endif                                   /* End of strload.*/

/*-------------------------------------------------------------------*
 * bit_pattern -- Change the \ddd format to a character symbol. It will *
 *        check to see if there are (at most two) other octal digits   *
 *        present.  It does not allow the return of the null character. *
 *        The pointer *ddd is only incremented by one for each extra   *
 *        digit, because the pointer will be incremented again upon    *
 *        returning from the function.                                 *
 *-------------------------------------------------------------------*/
bit_pattern(ddd)
char                    **ddd;
{
   extern char          *nullstr;
   int                  num;

   num = **ddd - '0';              /* Num is octal, otherwise we wouldn't be here.*/

   if (isoctal(*(*ddd + 1)))   /* Is the next character an octal digit?*/
   {
      num = 8 * num + *++*ddd - '0';

      if (isoctal(*(*ddd +1)))
         num = 8 * num + *++*ddd - '0';    /* How about this character?*/
   }

   if (!num)
      exit_error(10, nullstr);             /* No \0 allowed.*/

   return(num);

}
                                           /* End of bit_pattern.*/
/*-------------------------------------------------------------------*
 * attachf -- Take the lines of <file2>, if any, and attach them      *
 *        to the lines of <file1>, if any.  Slip determines how many  *
 *        lines of <file1> (<file2> if negative) are printed before   *
 *        printing the lines from both files together. It is possible to *
 *        specify some slippage even if the -a or -p flags are present. *
 *        This is not an error.  Attachf is smart enough to skip slip *
 *        in that case.                                               *
 *-------------------------------------------------------------------*/
attachf(fp1, fp2, slip, printsingle, printempty)
FILE                    *fp1, *fp2;
int                     slip;
Boolean                 printsingle, printempty;
{
   Boolean notempty;
   register int        nxtc;

   /* Handle slippage, if any, up to the end of the file.
    */
   for (; slip > 0 && fp1 != NULL; slip--)
   {
      notempty = (nxtc = nextc(fp1)) != '\n';

      if (nxtc == EOF)
      {
         fp1 = NULL;
         break;
      }

      put_line(fp1);

      if (printsingle && (printempty || notempty))
         printf(strf, bstring);

      putchar('\n');
   }

   if (slip < 0)
      slip = -slip;

   for (; slip > 0 && fp2 != NULL; slip--)
```

(continued on next page)

Listing One *(Listing continued, text begins on page 110.)*

```
      {
         if ((nxtc = nextc(fp2)) == EOF)
         {
            fp2 = NULL;
            break;
         }

         if (printsingle && (printempty || nxtc != '\n'))
            printf(strf, bstring);

         put_line(fp2);
         putchar('\n');
      }

   /* Paste the lines of each file together.
    */
   while (fpl != NULL && fp2 != NULL)
   {
      notempty = (nxtc = nextc(fpl)) != '\n';

      if (nxtc == EOF)
      {
         fpl = NULL;
         break;
      }

      put_line(fpl);

      if (printempty || notempty || nextc(fp2) != '\n')
         printf(strf, bstring);

      put_line(fp2);

      if (nextc(fp2) == EOF)
      {
         fp2 = NULL;
      }

      putchar('\n');
   }

   while (fpl != NULL)
   {
      notempty = (nxtc = nextc(fpl)) != '\n';
      put_line(fpl);

      if (nextc(fpl) == EOF)
         fpl = NULL;

      if (printsingle && (printempty || notempty ))
         printf(strf, bstring);

      putchar('\n');
   }

   while (fp2 != NULL)
   {
      if (printsingle && (printempty || nextc(fp2) != '\n'))
         printf(strf, bstring);

      put_line(fp2);

      if (nextc(fp2) == EOF)
         fp2 = NULL;

      putchar('\n');
   }
}                                          /* End of attachf.*/

/*-----------------------------------------------------------------*
 * put_line -- Get a line, print a line.                           *
 *-----------------------------------------------------------------*/
put_line(fp)
FILE            *fp;
{
   register int    c;

   while((c = getc(fp)) != '\n' && c != EOF)
      putchar(c);
}                                   /* End of put_line.*/

/*-----------------------------------------------------------------*
 * nextc -- What is the next character?  I realize that there are *
 *         some routines in some stdio.h files that do this for you, but *
 *         this is not true of all of them.  Hence this function.  *
 *-----------------------------------------------------------------*/
nextc(fp)
FILE            *fp;
{
   register int    c;

   c = getc(fp);
   ungetc(c, fp);

   return(c);
}                                          /* End of nextc.*/

/*-----------------------------------------------------------------*
 * exit_error -- Print out the appropriate error message for the   *
 *         appropriate error, then exit.                           *
 *-----------------------------------------------------------------*/
exit_error(errcode, details)
int             errcode;
char            *details;
{
   extern char     *error_msg[];

   fprintf(stderr, error_msg[errcode], program_name, details);
   exit (1);
}                                          /* End of exit_error.*/
```

End Listing

Listing One *(Text begins on page 120.)*

```
Listing 1. CHANGE.BAS Utility to search/replace text in a number of files.

1000 ' Batch Find/Replace Utility  Version 1.0  10/29/86
1005 ' IBM PC                   BASICA version 2 or later
1010 ' Copyright (c) 1987  Namir Clement Shammas
1020 DEFINT A-Z
1030 DIM FILENAME$(20),STRNG$(30),REPLACE(30),REPLACE$(30),L$(500)
1040 TRUE  = 1
1050 FALSE = 0
1060 MAX.LINES = 500 ' Current maximum number of lines read from a file
1900 CLS
1910 T$ = "BATCH FILE FIND/REPLACE PROGRAM" : GOSUB 8000
1920 PRINT
1930 T$ = "VERSION 1.0" : GOSUB 8000
1940 PRINT : PRINT
2000 GOSUB 5000 ' Get filenames
2010 GOSUB 6000 ' Get strings
2030 FOR IFILE = 1 TO NUM.FILES
2060     GOSUB 7000 ' Read text lines from file
2070     FOR I = 1 TO NUM.STRINGS
2080         FOUND = FALSE
2090         FOR J = 1 TO NUM.LINES
2100             PTR = INSTR(L$(J),STRNG$(I))
2110             WHILE PTR > 0
2120                 IF (FOUND = TRUE) THEN 2150
2130                     FOUND = TRUE
2140                     LPRINT "KEYWORD : ";STRNG$(I)
2150                 B$ = STR$(J) + ":"
2153                 OFFSET = LEN(B$)
2155                 LPRINT J;":";L$(J)
2160                 LPRINT SPC(PTR+OFFSET);"^"
2170                 IF (REPLACE(I) = FALSE) THEN 2240
2180                     FIRST$ = ""
2190                     IF PTR > 1 THEN FIRST$ = MID$(L$(J),1,(PTR-1))
2200                     LAST$ = ""
2210                     IF (PTR+LEN(STRNG$(I))) => LEN(L$(J)) THEN 2230
2220                         LAST$ = MID$(L$(J),(PTR+LEN(STRNG$(I))))
2230                     L$(J) = FIRST$ + REPLACE$(I) + LAST$
2231                     LPRINT "BECOMES" : LPRINT
2233                     LPRINT J;":";L$(J) : LPRINT : LPRINT
2240                 PTR = INSTR(PTR+1,L$(J),STRNG$(I))
2250             WEND
2260         NEXT J
2270     NEXT I
2275     GOSUB 9000 ' Write file back
2277     LPRINT : LPRINT
2280 NEXT IFILE
2290 LPRINT CHR$(140) ' FORM FEED
3000 END '-----------------------------------------------------------
5000 ' Subroutine to input filenames from the keyboard
5010 NUM.FILES = 0
5020 WHILE NUM.FILES <= 0
5030     INPUT "Enter number of files ";NUM.FILES
5040     PRINT
5050 WEND
5060 FOR I = 1 TO NUM.FILES
5070    PRINT "Enter filename # ";I;" ";
5080    INPUT FILENAME$(I) : PRINT
5090    IF FILENAME$(I) = "" THEN 5070
5100 NEXT I
5110 RETURN
6000 ' Subroutines to inpur search/replace strings
6010 NUM.STRINGS = 0
6020 WHILE NUM.STRINGS <= 0
6030     INPUT "Enter number of search/replace strings ";NUM.STRINGS
6040     PRINT
6050 WEND
6060 FOR I = 1 TO NUM.STRINGS
6065     REPLACE$(I) = ""
6070     PRINT : PRINT "For string # ";I
6080     INPUT "    Enter string ";STRNG$(I)
6090     INPUT "    R)eplace F)ind ";A$
6100     IF (INSTR("Rr",MID$(A$,1,1)) = 0) THEN REPLACE(I) =
         FALSE ELSE REPLACE(I) = TRUE
6110     IF REPLACE(I) = FALSE THEN 6125
6120     INPUT "  · Enter replacement string ";REPLACE$(I)
6125 PRINT
6130 NEXT I
6140 RETURN
7000 ' Subroutines to read text lines
7003 LPRINT "PROCESSING FILE : ";FILENAME$(IFILE)
```

```
7006 OPEN "I",1,FILENAME$(IFILE)
7010 NUM.LINES = 0
7020 WHILE (NOT EOF(1)) AND (NUM.LINES <= MAX.LINES)
7030     NUM.LINES = NUM.LINES + 1
7040     LINE INPUT#1,L$(NUM.LINES)
7050 WEND
7060 CLOSE #1
7070 RETURN
8000 ' Subroutine to center a message
8010 PRINT SPC(40 - LEN(T$)/2);T$
8020 RETURN
9000 'Subroutine to write the updated file
9010 OPEN "O",1,FILENAME$(IFILE)
9020 FOR I = 1 TO NUM.LINES
9030     PRINT#1,L$(I)
9040 NEXT I
9050 CLOSE#1
9060 RETURN
```

End Listing One

Listing Two

Listing 2. CHNG1.TRU the version of True BASIC CHANGE.BAS produced by the BASIC-Converter.

```
10 ! This program converted from the Microsoft Advanced Basic
11 ! language on the IBM PC to the True BASIC language.
12 !
13 ! Convertor copyright (c) 1985 by:
14 !      True BASIC, Inc.
15 !      Hanover, NH  03755
16 !      All rights reserved.
17 !
18 ! True BASIC makes no warranty, expressed or implied, that
19 ! this converted program is a precise and accurate equivalent
20 ! of the original BasicA program.  This conversion is provided
21 ! only as an aid to a complete conversion by the owner of the
22 ! program being converted.
23 !
24 LIBRARY "deflib"
25 DECLARE DEF csrlin, oef, fre, hex$, inkey$, loc, lof
26 DECLARE DEF mki$, mks$, cvi, cvs, oct$, csr_pos, val_a, err, erl
27
28 DEF Eof (f)
29    IF end #f then LET eof = -1 else LET eof = 0
30 END DEF
31
32 DEF Loc (f)
33    ASK #f: record T_ARG1
34    LET t_arg1 = -int(-(t_arg1-1)/128)
35    IF t_arg1 = 0 then let loc = 1 else let loc = t_arg1
36 END DEF
37
38 DEF Lof (f)
39    ASK #f: filesize T_ARG1
40    LET lof = t_arg1
41 END DEF
42
43 OPTION BASE 0
44
1000 ! Batch Find/Replace Utility  Version 1.0  10/29/86
1005 ! IBM PC                      BASICA version 2 or later
1010 ! Copyright (c) 1987  Namir Clement Shammas
1020 !  defint A-Z
1030 dim filename$(20), strng$(30), replace(30), replace$(30), l$(500)
1040 let true = 1
1050 let false = 0
1060 let max__lines = 500   ! Current maximum number of lines read from a file
1900 clear
1910 let t$ = "BATCH FILE FIND/REPLACE PROGRAM"
1911 gosub 8000
1920 print
1930 let t$ = "VERSION 1.0"
1931 gosub 8000
1940 print
1941 print
1945 OPEN #9 : PRINTER
2000 gosub 5000   ! Get filenames
2010 gosub 6000   ! Get strings
2030 for ifile = 1 to num__files
2060 gosub 7000   ! Read text lines from file
2070 for i = 1 to num__strings
```

(continued on next page)

Listing Two *(Listing continued, text begins on page 120.)*

```
2080 let found = false
2090 for j = 1 to num__lines
2100 let ptr = pos(l$(j),strng$(i))
2110 do while ptr > 0
2120 if (found = true) then goto 2150
2130 let found = true
2140 print #9 : "KEYWORD : ";STRNG$(I)
2150 let b$ = str$(j) & ":"
2153 let offset = round(len(b$))
2155 print #9 :  J;":";L$(J)
2160 print #9 :  REPEAT$(" ",(PTR+OFFSET+1));"^" ! Manual fix on this line
2170 if (replace(i) = false) then goto 2240
2180 let first$ = ""
2190 if ptr > 1 then let first$ = (l$(j))[1:1+(ptr-1)-1]
2200 let last$ = ""
2210 if (ptr+len(strng$(i))) => len(l$(j)) then goto 2230
2220 let last$ = (l$(j))[(ptr+len(strng$(i))):maxnum]
2230 let l$(j) = first$ & replace$(i) & last$
2231 print #9 :  "BECOMES"
2232 print #9 :
2233 print #9 :  J;":";L$(J)
2234 print #9 :
2235 print #9 :
2240 let ptr = pos(l$(j),strng$(i),ptr+1)
2250 loop
2260 next j
2270 next i
2275 gosub 9000   ! Write file back
2277 print #9 :
2278 print #9 :
2280 next ifile
2290 print #9 :  CHR$(140)    ! FORM FEED
3000 stop   !---------------------------------------------------------
5000 ! Subroutine to input filenames from the keyboard
5010 let num__files = 0
5020 do while num__files <= 0
5030 input prompt "Enter number of files ": num__files
5040 print
5050 loop
5060 for i = 1 to num__files
5070 print "Enter filename # "; i; " ";
5080 input filename$(i)
5081 print
5090 if filename$(i) = "" then goto 5070
5100 next i
5110 return
6000 ! Subroutines to inpur search/replace strings
6010 let num__strings = 0
6020 do while num__strings <= 0
6030 input prompt "Enter number of search/replace strings ": num__strings
6040 print
6050 loop
6060 for i = 1 to num__strings
6065 let replace$(i) = ""
6070 print
6071 print "For string # "; i
6080 input prompt "    Enter string ": strng$(i)
6090 input prompt "    R)eplace F)ind ": a$
6100 if (pos("Rr",(a$)[1:1]) = 0) then let replace(i) =
     false else let replace(i) = true
6110 if replace(i) = false then goto 6125
6120 input prompt "    Enter replacement string ": replace$(i)
6125 print
6130 next i
6140 return
7000 ! Subroutines to read text lines
7003 print #9 :  "PROCESSING FILE : ";FILENAME$(IFILE)
7006 open #1: name filename$(ifile), access input, create old
7010 let num__lines = 0
7020 do while ((not eof(1) <> 0)) and (num__lines <= max__lines)
7030 let num__lines = num__lines+1
7040 line input #1:l$(num__lines) ! Manual fix here
7050 loop
7060 close  #1
7070 return
8000 ! Subruotine to center a message
8010 print tab(csr_pos+40-len(t$)/2); t$
8020 return
9000 !Subroutine to write the updated file
9010 open #1: name filename$(ifile), access output, create old
9015 erase #1 ! this line is added
```

Listing One *(Text begins on page 120.)*

```
Listing 1. CHANGE.BAS Utility to search/replace text in a number of files.

1000 ' Batch Find/Replace Utility  Version 1.0  10/29/86
1005 ' IBM PC                   BASICA version 2 or later
1010 ' Copyright (c) 1987  Namir Clement Shammas
1020 DEFINT A-Z
1030 DIM FILENAME$(20),STRNG$(30),REPLACE(30),REPLACE$(30),L$(500)
1040 TRUE  = 1
1050 FALSE = 0
1060 MAX.LINES = 500 ' Current maximum number of lines read from a file
1900 CLS
1910 T$ = "BATCH FILE FIND/REPLACE PROGRAM" : GOSUB 8000
1920 PRINT
1930 T$ = "VERSION 1.0" : GOSUB 8000
1940 PRINT : PRINT
2000 GOSUB 5000 ' Get filenames
2010 GOSUB 6000 ' Get strings
2030 FOR IFILE = 1 TO NUM.FILES
2060     GOSUB 7000 ' Read text lines from file
2070     FOR I = 1 TO NUM.STRINGS
2080         FOUND = FALSE
2090         FOR J = 1 TO NUM.LINES
2100             PTR = INSTR(L$(J),STRNG$(I))
2110             WHILE PTR > 0
2120                 IF (FOUND = TRUE) THEN 2150
2130                     FOUND = TRUE
2140                     LPRINT "KEYWORD : ";STRNG$(I)
2150                 B$ = STR$(J) + ":"
2153                 OFFSET = LEN(B$)
2155                 LPRINT J;":";L$(J)
2160                 LPRINT SPC(PTR+OFFSET);"^"
2170                 IF (REPLACE(I) = FALSE) THEN 2240
2180                     FIRST$ = ""
2190                     IF PTR > 1 THEN FIRST$ = MID$(L$(J),1,(PTR-1))
2200                     LAST$ = ""
2210                     IF (PTR+LEN(STRNG$(I))) => LEN(L$(J)) THEN 2230
2220                         LAST$ = MID$(L$(J),(PTR+LEN(STRNG$(I))))
2230                     L$(J) = FIRST$ + REPLACE$(I) + LAST$
2231                     LPRINT "BECOMES" : LPRINT
2233                     LPRINT J;":";L$(J) : LPRINT : LPRINT
2240                 PTR = INSTR(PTR+1,L$(J),STRNG$(I))
2250             WEND
2260         NEXT J
2270     NEXT I
2275     GOSUB 9000 ' Write file back
2277     LPRINT : LPRINT
2280 NEXT IFILE
2290 LPRINT CHR$(140) ' FORM FEED
3000 END '----------------------------------------------------------
5000 ' Subroutine to input filenames from the keyboard
5010 NUM.FILES = 0
5020 WHILE NUM.FILES <= 0
5030     INPUT "Enter number of files ";NUM.FILES
5040     PRINT
5050 WEND
5060 FOR I = 1 TO NUM.FILES
5070     PRINT "Enter filename # ";I;" ";
5080     INPUT FILENAME$(I) : PRINT
5090     IF FILENAME$(I) = "" THEN 5070
5100 NEXT I
5110 RETURN
6000 ' Subroutines to inpur search/replace strings
6010 NUM.STRINGS = 0
6020 WHILE NUM.STRINGS <= 0
6030     INPUT "Enter number of search/replace strings ";NUM.STRINGS
6040     PRINT
6050 WEND
6060 FOR I = 1 TO NUM.STRINGS
6065     REPLACE$(I) = ""
6070     PRINT : PRINT "For string # ";I
6080     INPUT "    Enter string ";STRNG$(I)
6090     INPUT "    R)eplace F)ind ";A$
6100     IF (INSTR("Rr",MID$(A$,1,1)) = 0) THEN REPLACE(I) =
         FALSE ELSE REPLACE(I) = TRUE
6110     IF REPLACE(I) = FALSE THEN 6125
6120     INPUT "  · Enter replacement string ";REPLACE$(I)
6125 PRINT
6130 NEXT I
6140 RETURN
7000 ' Subroutines to read text lines
7003 LPRINT "PROCESSING FILE : ";FILENAME$(IFILE)
```

```
7006 OPEN "I",1,FILENAMES$(IFILE)
7010 NUM.LINES = 0
7020 WHILE (NOT EOF(1)) AND (NUM.LINES <= MAX.LINES)
7030    NUM.LINES = NUM.LINES + 1
7040    LINE INPUT#1,L$(NUM.LINES)
7050 WEND
7060 CLOSE #1
7070 RETURN
8000 ' Subroutine to center a message
8010 PRINT SPC(40 - LEN(T$)/2);T$
8020 RETURN
9000 'Subroutine to write the updated file
9010 OPEN "O",1,FILENAMES$(IFILE)
9020 FOR I = 1 TO NUM.LINES
9030    PRINT#1,L$(I)
9040 NEXT I
9050 CLOSE#1
9060 RETURN
```

End Listing One

Listing Two

Listing 2. CHNG1.TRU the version of True BASIC CHANGE.BAS produced by the BASIC-Converter.

```
10 ! This program converted from the Microsoft Advanced Basic
11 ! language on the IBM PC to the True BASIC language.
12 !
13 ! Convertor copyright (c) 1985 by:
14 !      True BASIC, Inc.
15 !      Hanover, NH  03755
16 !      All rights reserved.
17 !
18 ! True BASIC makes no warranty, expressed or implied, that
19 ! this converted program is a precise and accurate equivalent
20 ! of the original BasicA program.  This conversion is provided
21 ! only as an aid to a complete conversion by the owner of the
22 ! program being converted.
23 !
24 LIBRARY "deflib"
25 DECLARE DEF csrlin, oef, fre, hex$, inkey$, loc, lof
26 DECLARE DEF mki$, mks$, cvi, cvs, oct$, csr_pos, val_a, err, erl
27
28 DEF Eof (f)
29    IF end #f then LET eof = -1 else LET eof = 0
30 END DEF
31
32 DEF Loc (f)
33    ASK #f: record T_ARG1
34    LET t_arg1 = -int(-(t_arg1-1)/128)
35    IF t_arg1 = 0 then let loc = 1 else let loc = t_arg1
36 END DEF
37
38 DEF Lof (f)
39    ASK #f: filesize T_ARG1
40    LET lof = t_arg1
41 END DEF
42
43 OPTION BASE 0
44
1000 ! Batch Find/Replace Utility  Version 1.0  10/29/86
1005 ! IBM PC                     BASICA version 2 or later
1010 ! Copyright (c) 1987  Namir Clement Shammas
1020 !  defint A-Z
1030 dim filename$(20), strng$(30), replace(30), replace$(30), l$(500)
1040 let true = 1
1050 let false = 0
1060 let max__lines = 500   ! Current maximum number of lines read from a file
1900 clear
1910 let t$ = "BATCH FILE FIND/REPLACE PROGRAM"
1911 gosub 8000
1920 print
1930 let t$ = "VERSION 1.0"
1931 gosub 8000
1940 print
1941 print
1945 OPEN #9 : PRINTER
2000 gosub 5000    ! Get filenames
2010 gosub 6000    ! Get strings
2030 for ifile = 1 to num__files
2060 gosub 7000    ! Read text lines from file
2070 for i = 1 to num__strings
```

(continued on next page)

Listing Two *(Listing continued, text begins on page 120.)*

```
2080 let found = false
2090 for j = 1 to num__lines
2100 let ptr = pos(l$(j),strng$(i))
2110 do while ptr > 0
2120 if (found = true) then goto 2150
2130 let found = true
2140 print #9 : "KEYWORD : ";STRNG$(I)
2150 let b$ = str$(j) & ":"
2153 let offset = round(len(b$))
2155 print #9 :  J;":";L$(J)
2160 print #9 :  REPEAT$(" ",(PTR+OFFSET+1));"^" ! Manual fix on this line
2170 if (replace(i) = false) then goto 2240
2180 let first$ = ""
2190 if ptr > 1 then let first$ = (l$(j))[1:1+(ptr-1)-1]
2200 let last$ = ""
2210 if (ptr+len(strng$(i))) => len(l$(j)) then goto 2230
2220 let last$ = (l$(j))[(ptr+len(strng$(i))):maxnum]
2230 let l$(j) = first$ & replace$(i) & last$
2231 print #9 :  "BECOMES"
2232 print #9 :
2233 print #9 :  J;":";L$(J)
2234 print #9 :
2235 print #9 :
2240 let ptr = pos(l$(j),strng$(i),ptr+1)
2250 loop
2260 next j
2270 next i
2275 gosub 9000   ! Write file back
2277 print #9 :
2278 print #9 :
2280 next ifile
2290 print #9 :  CHR$(140)   ! FORM FEED
3000 stop   !----------------------------------------------------
5000 ! Subroutine to input filenames from the keyboard
5010 let num__files = 0
5020 do while num__files <= 0
5030 input prompt "Enter number of files ": num__files
5040 print
5050 loop
5060 for i = 1 to num__files
5070 print "Enter filename # "; i; " ";
5080 input filename$(i)
5081 print
5090 if filename$(i) = "" then goto 5070
5100 next i
5110 return
6000 ! Subroutines to inpur search/replace strings
6010 let num__strings = 0
6020 do while num__strings <= 0
6030 input prompt "Enter number of search/replace strings ": num__strings
6040 print
6050 loop
6060 for i = 1 to num__strings
6065 let replace$(i) = ""
6070 print
6071 print "For string # "; i
6080 input prompt "    Enter string ": strng$(i)
6090 input prompt "    R)eplace F)ind ": a$
6100 if (pos("Rr",(a$)[1:1]) = 0) then let replace(i) =
     false else let replace(i) = true
6110 if replace(i) = false then goto 6125
6120 input prompt "    Enter replacement string ": replace$(i)
6125 print
6130 next i
6140 return
7000 ! Subroutines to read text lines
7003 print #9 :  "PROCESSING FILE : ";FILENAME$(IFILE)
7006 open #1: name filename$(ifile), access input, create old
7010 let num__lines = 0
7020 do while ((not eof(1) <> 0)) and (num__lines <= max__lines)
7030 let num__lines = num__lines+1
7040 line input #1:l$(num__lines) ! Manual fix here
7050 loop
7060 close  #1
7070 return
8000 ! Subruotine to center a message
8010 print tab(csr_pos+40-len(t$)/2); t$
8020 return
9000 !Subroutine to write the updated file
9010 open #1: name filename$(ifile), access output, create old
9015 erase #1 ! this line is added
```

```
9020 for i = 1 to num__lines
9030 print #1:l$(i)
9040 next i
9050 close #1
9060 return
9061 end
```

End Listing Two

Listing Three

```
Listing 3. CHNG2.TRU the True BASIC version of CHANGE.BAS that is
translated manually.

! Batch Find/Replace Utility  Version 1.0  10/29/86
! IBM PC                  True BASIC version 1
! Copyright  (c) 1987  Namir Clement Shammas
DIM FILENAME$(20),STRNG$(30),REPLACE(30),REPLACE$(30),L$(500)
LET TRUE  = 1
LET FALSE = 0
LET MAX_LINES = 500 ! Current maximum number of lines read from a file
CLEAR ! Clear screen
CALL CenterText("BATCH FILE FIND/REPLACE PROGRAM")
PRINT
CALL CenterText("VERSION 1.0")
PRINT
PRINT
OPEN #9 : PRINTER
CALL GetFile(FILENAME$, NUM_FILES) ! Get filenames
CALL GetStrings(STRNG$,REPLACE$,REPLACE,NUM_STRINGS) ! Get strings
FOR IFILE = 1 TO NUM_FILES
   CALL ReadLines(L$,FILENAME$,IFILE, NUM_LINES) ! Read text lines from file
   FOR I = 1 TO NUM_STRINGS
      LET FOUND = FALSE
      FOR J = 1 TO NUM_LINES
         LET PTR = POS(L$(J),STRNG$(I))
         DO WHILE PTR > 0
           IF (FOUND = FALSE) THEN
              LET FOUND = TRUE
              PRINT #9 :  "KEYWORD : ";STRNG$(I)
           END IF
           LET B$ = STR$(J) & ":" ! Use & to concatenate strings
           LET OFFSET = LEN(B$)
           PRINT #9 :  J;":";L$(J)
           PRINT #9 :  REPEAT$(" ",(PTR+OFFSET+1));"^"
           IF (REPLACE(I) = TRUE) THEN
              LET FIRST$ = ""
              IF PTR > 1 THEN LET FIRST$ = L$(J)[1:(PTR-1)]
              LET LAST$ = ""
              IF (PTR+LEN(STRNG$(I))) < LEN(L$(J)) THEN
                 LET LAST$ = L$(J)[(PTR+LEN(STRNG$(I))):LEN(L$(J))]
              END IF
              LET L$(J) = FIRST$ & REPLACE$(I) & LAST$
              PRINT #9 :  "BECOMES"
              PRINT #9 :
              PRINT #9 :  J;":";L$(J)
              PRINT #9 :
              PRINT #9 :
           END IF
           LET PTR = POS(L$(J),STRNG$(I),(PTR+1))
         LOOP
      NEXT J
   NEXT I
   CALL WriteLines(L$,FILENAME$,REPLACE,IFILE,NUM_LINES)
   ! Write file back
   PRINT #9 :
   PRINT #9 :
NEXT IFILE
PRINT #9 : CHR$(140) ! FORM FEED

SUB GetFile(FILENAME$(), NUM_FILES)
! Subroutine to input filenames from the keyboard
LET NUM_FILES = 0
DO WHILE NUM_FILES <= 0
   INPUT PROMPT "Enter number of files ":NUM_FILES
   PRINT
LOOP
FOR I = 1 TO NUM_FILES
   LET FILENAME$(I) = ""
   DO WHILE FILENAME$(I) = ""
      PRINT "Enter filename # ";I;" ";
      INPUT FILENAME$(I)
```

(continued on next page)

Listing Three *(Listing continued, text begins on page 120.)*

```
      PRINT
   LOOP
NEXT I
END SUB

SUB GetStrings(STRNGS$(),REPLACES$(),REPLACE(),NUM_STRINGS)

! Subroutines to inpur search/replace strings
LET NUM_STRINGS = 0
DO WHILE NUM_STRINGS <= 0
   INPUT PROMPT "Enter number of search/replace strings ":NUM_STRINGS
   PRINT
LOOP
FOR I = 1 TO NUM_STRINGS
   LET REPLACES$(I) = ""
   PRINT
   PRINT "For string # ";I
   INPUT PROMPT "    Enter string ":STRNGS$(I)
   INPUT PROMPT "    R)eplace F)ind ":A$
   IF (POS("Rr",A$[1:1]) = 0) THEN
      LET REPLACE(I) = FALSE
   ELSE
      LET REPLACE(I) = TRUE
      INPUT PROMPT "    Enter replacement string ":REPLACES$(I)
   END IF
   PRINT
NEXT I
END SUB

SUB ReadLines(L$(),FILENAMES$(),INDEX,NUM_LINES)
! Subroutines to read text lines
PRINT #9 :  "PROCESSING FILE : ";FILENAMES$(INDEX)
OPEN #1 : NAME FILENAMES$(INDEX), ORGANIZATION TEXT, ACCESS INPUT, CREATE OLD
LET NUM_LINES = 0
DO WHILE MORE #1
   LET NUM_LINES = NUM_LINES + 1
   LINE INPUT#1 : L$(NUM_LINES)
LOOP
CLOSE #1
END SUB

SUB CenterText(T$)
! Subroutine to center a message
PRINT REPEAT$(" ",(40 - LEN(T$)/2));T$
END SUB

SUB WriteLines(L$(),FILENAMES$(),INDEX,NUM_LINES)
!Subroutine to write the updated file
OPEN #1 : NAME FILENAMES$(INDEX), ORGANIZATION TEXT, ACCESS OUTPUT, CREATE OLD
ERASE #1
FOR I = 1 TO NUM_LINES
   PRINT#1 : L$(I)
NEXT I
CLOSE#1
END SUB

END
```

End Listing Three

Listing Four

```
Listing 4. CHNG1.BAS the first QuickBASIC version of CHANGE.BAS that is
translated manually.

' Batch Find/Replace Utility  Version 1.0  10/29/86
' IBM PC                   QuickBASIC version 2
' Copyright (c) 1987  Namir Clement Shammas
DEFINT A-Z
DIM FILENAMES$(20),STRNGS$(30),REPLACE(30),REPLACES$(30),L$(500)
TRUE  = 1
FALSE = 0
MAX.LINES = 500 ' Current maximum number of lines read from a file
CLS
T$ = "BATCH FILE FIND/REPLACE PROGRAM" : GOSUB Center
PRINT
T$ = "VERSION 1.0" : GOSUB Center
     PRINT : PRINT
     GOSUB GetFile ' Get filenames
     GOSUB GetStrings ' Get strings
     FOR IFILE = 1 TO NUM.FILES
```

```
        GOSUB ReadLines ' Read text lines from file
        FOR I = 1 TO NUM.STRINGS
            FOUND = FALSE
            FOR J = 1 TO NUM.LINES
                PTR = INSTR(L$(J),STRNG$(I))
                WHILE PTR > 0
                      IF (FOUND = FALSE) THEN
                        FOUND = TRUE
                           LPRINT "KEYWORD : ";STRNG$(I)
                      END IF
                      B$ = STR$(J) + ":"
                    OFFSET = LEN(B$)
                    LPRINT J;":";L$(J)
                    LPRINT SPC(PTR+OFFSET);"^"
                      IF (REPLACE(I) = TRUE) THEN
                           FIRST$ = ""
                        IF PTR > 1 THEN FIRST$ = MID$(L$(J),1,(PTR-1))
                        LAST$ = ""
                           IF (PTR+LEN(STRNG$(I))) < LEN(L$(J)) THEN
                               LAST$ = MID$(L$(J),(PTR+LEN(STRNG$(I))))
                           END IF
                           L$(J) = FIRST$ + REPLACE$(I) + LAST$
                        LPRINT "BECOMES" : LPRINT
                           LPRINT J;":";L$(J) : LPRINT : LPRINT
                      END IF
                      PTR = INSTR(PTR+1,L$(J),STRNG$(I))
                WEND
            NEXT J
        NEXT I
         GOSUB WriteLines ' Write file back
        LPRINT : LPRINT
    NEXT IFILE
    LPRINT CHR$(140) ' FORM FEED
    END '----------------------------------------------------------
GetFile: ' Subroutine to input filenames from the keyboard
    NUM.FILES = 0
    WHILE NUM.FILES <= 0
        INPUT "Enter number of files ";NUM.FILES
        PRINT
    WEND
    FOR I = 1 TO NUM.FILES
        FILENAME$(I) = ""
        WHILE FILENAME$(I) = ""
           PRINT "Enter filename # ";I;" ";
           INPUT FILENAME$(I) : PRINT
        WEND
    NEXT I
    RETURN
GetStrings: ' Subroutines to inpur search/replace strings
    NUM.STRINGS = 0
    WHILE NUM.STRINGS <= 0
        INPUT "Enter number of search/replace strings ";NUM.STRINGS
        PRINT
    WEND
    FOR I = 1 TO NUM.STRINGS
        REPLACE$(I) = ""
        PRINT : PRINT "For string # ";I
        INPUT "    Enter string ";STRNG$(I)
        INPUT "    R)eplace F)ind ";A$
        IF (INSTR("Rr",MID$(A$,1,1)) = 0) THEN REPLACE(I) =
        FALSE ELSE REPLACE(I) = TRUE
        IF REPLACE(I) = TRUE THEN
           INPUT "    Enter replacement string ";REPLACE$(I)
        END IF
        PRINT
    NEXT I
    RETURN
ReadLines: ' Subroutines to read text lines
    LPRINT "PROCESSING FILE : ";FILENAME$(IFILE)
    OPEN "I",1,FILENAME$(IFILE)
    NUM.LINES = 0
    WHILE (NOT EOF(1)) AND (NUM.LINES <= MAX.LINES)
       NUM.LINES = NUM.LINES + 1
       LINE INPUT#1,L$(NUM.LINES)
    WEND
    CLOSE #1
    RETURN
Center:  ' Subroutine to center a message
    PRINT SPC(40 - LEN(T$)/2);T$
    RETURN
WriteLines: 'Subroutine to write the updated file
    OPEN "O",1,FILENAME$(IFILE)
```

(continued on next page)

Listing Four *(Listing continued, text begins on page 120.)*

```
    FOR I = 1 TO NUM.LINES
        PRINT#1,L$(I)
    NEXT I
    CLOSE#1
    RETURN
```

End Listing Four

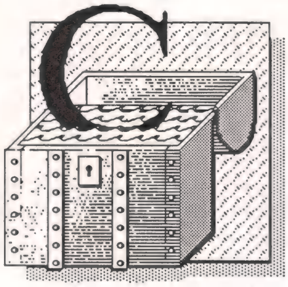
Listing Five

```
Listing 5. CHNG2.BAS the second QuickBASIC version of CHANGE.BAS that is
translated manually.

' Batch Find/Replace Utility  Version 1.0  10/29/86
' IBM PC                   QuickBASIC version 2
' Copyright (c) 1987  Namir Clement Shammas
DEFINT A-Z
DIM FILENAME$(20),STRNG$(30),REPLACE(30),REPLACE$(30),L$(500)
TRUE  = 1
FALSE = 0
MAX.LINES = 500 ' Current maximum number of lines read from a file
CLS
CALL CenterText("BATCH FILE FIND/REPLACE PROGRAM")
PRINT
CALL CenterText("VERSION 1.0")
PRINT : PRINT
CALL GetFile(FILENAME$(), NUM.FILES) ' Get filenames
CALL GetStrings(STRNG$(),REPLACE$(),REPLACE(),NUM.STRINGS) ' Get strings
FOR IFILE = 1 TO NUM.FILES
   ' Read text lines from file
   CALL ReadLines(L$(),FILENAME$(),IFILE, NUM.LINES)
   FOR I = 1 TO NUM.STRINGS
      FOUND = FALSE
      FOR J = 1 TO NUM.LINES
          PTR = INSTR(L$(J),STRNG$(I))
          WHILE PTR > 0
            IF (FOUND = FALSE) THEN
               FOUND = TRUE
               LPRINT "KEYWORD : ";STRNG$(I)
            END IF
            B$ = STR$(J) + ":"
            OFFSET = LEN(B$)
            LPRINT J;":";L$(J)
            LPRINT SPC(PTR+OFFSET);"^"
            IF (REPLACE(I) = TRUE) THEN
               FIRST$ = ""
               IF PTR > 1 THEN FIRST$ = MID$(L$(J),1,(PTR-1))
               LAST$ = ""
               IF (PTR+LEN(STRNG$(I))) < LEN(L$(J)) THEN
                   LAST$ = MID$(L$(J),(PTR+LEN(STRNG$(I))))
               END IF
               L$(J) = FIRST$ + REPLACE$(I) + LAST$
               LPRINT "BECOMES" : LPRINT
               LPRINT J;":";L$(J) : LPRINT : LPRINT
            END IF
            PTR = INSTR(PTR+1,L$(J),STRNG$(I))
          WEND
      NEXT J
   NEXT I
   ' Write file back
   CALL WriteLines(L$(),FILENAME$(),REPLACE(),IFILE,NUM.LINES)
   LPRINT : LPRINT
NEXT IFILE
LPRINT CHR$(140) ' FORM FEED
END '----------------------------------------------------------

SUB GetFile(FILENAME$(1), NUM.FILES) STATIC
' Subroutine to input filenames from the keyboard
     NUM.FILES = 0
     WHILE NUM.FILES <= 0
         INPUT "Enter number of files ";NUM.FILES
         PRINT
     WEND
     FOR I = 1 TO NUM.FILES
         FILENAME$(I) = ""
         WHILE FILENAME$(I) = ""
            PRINT "Enter filename # ";I;" ";
            INPUT FILENAME$(I) : PRINT
         WEND
     NEXT I
END SUB
```

```
SUB GetStrings(STRNG$(1),REPLACE$(1),REPLACE(1),NUM.STRINGS) STATIC
' Subroutines to inpur search/replace strings
     NUM.STRINGS = 0
     WHILE NUM.STRINGS <= 0
         INPUT "Enter number of search/replace strings ";NUM.STRINGS
         PRINT
     WEND
     FOR I = 1 TO NUM.STRINGS
         REPLACE$(I) = ""
         PRINT : PRINT "For string # ";I
         INPUT "    Enter string ";STRNG$(I)
         INPUT "    R)eplace F)ind ";A$
          IF (INSTR("Rr",MID$(A$,1,1)) = 0) THEN

             REPLACE(I) = FALSE
          ELSE
             REPLACE(I) = TRUE
             INPUT "    Enter replacement string ";REPLACE$(I)
          END IF
          PRINT
     NEXT I
END SUB

SUB ReadLines(L$(1),FILENAME$(1),INDEX,NUM.LINES) STATIC
' Subroutines to read text lines
     LPRINT "PROCESSING FILE : ";FILENAME$(INDEX)
     OPEN "I",1,FILENAME$(INDEX)
     NUM.LINES = 0
     WHILE (NOT EOF(1)) ' AND (NUM.LINES <= MAX.LINES)
        NUM.LINES = NUM.LINES + 1
        LINE INPUT#1,L$(NUM.LINES)
     WEND
     CLOSE #1
END SUB

SUB CenterText(T$) STATIC
' Subroutine to center a message
     PRINT SPC(40 - LEN(T$)/2);T$
END SUB

SUB WriteLines(L$(1),FILENAME$(1),INDEX,NUM.LINES) STATIC
' Subroutine to write the updated file
     OPEN "O",1,FILENAME$(INDEX)
     FOR I = 1 TO NUM.LINES
        PRINT#1,L$(I)
     NEXT I
     CLOSE#1
END SUB
```

End Listings

Nr: A C Implementation of Nroff, Part 2



This month I'll continue discussing the nr text formatter that I introduced last month. I'll present the first part of a complete users' guide and continue it in the next column. The source-code disk contains a complete implementation of the ms macro package. (See the end of this column for information about the source code disk.) Nr is as much a programming language as it is a text formatter, and a look at a complex macro package such as ms can show you how to program in that language.

I should preface this article by saying that I've implemented the Unix nroff as closely as I can. Over the course of several years, I've learned more things about nroff than I actually care to know. I do not claim, however, to know everything there is to know about the real nroff. As a consequence, there may be a few differences between nr and the real nroff, introduced because I can't figure out how the real nroff works. Sorry. I should also say that, though I've used the program presented here for several years now and don't know about any bugs, I'm a creature of habit and probably haven't exercised those parts of the program that have bugs in them. In particular, when you get into the realm of fancy laser printers and proportional spacing, nr may not work without your having to modify the program somewhat. It works fine on the various printers I own (HP Thinkjet, Brother HR-15, and HP-Laserjet+), but these are the only print-

by Allen Holub

ers on which the program has been used. I've added the proportional-spacing features very recently, so I don't have as much confidence in that part of the program as I do in the older parts.

As I write this article, I'm looking at the code more closely than I have for a while. As a consequence I'm noticing (and fixing) a few nroff incompatabilities I hadn't noticed before. Two such fixes affect one of the subroutines I discussed last month—the expression parser in parse.c. This parser treats the *'str1'str2'* expression as if it were using the C *strcmp( )* function. Nroff, on the other hand, evaluates this expression to true if the two strings are equal and to false if they are not—the opposite of *strcmp( )*. A second problem with the parser is actually a bug. It shouldn't recognize a quote as white space. To modify parse.c to fix these problems, replace line 293 of parse.c with:

```
rval = !strcmp( s1, s2 );
```

and change line 137 to:

```
while( isspace(*Str) )
```

Nr Users' Guide

It's almost impossible to describe a program as complex as nr in an orderly fashion because there's no way to organize the material to avoid forward references. Consequently, you'll probably have to read this guide (and its conclusion in my next column) twice—once to get a general idea of how the program works and a second time to fill in the details.

Nr is an almost complete implementation of the Unix nroff text formatter. It incorporates several of troff's functions as well, and it can generate output for most printers without any modifications to the source code.

Nr is a compiler-like text formatter. You create the input text with a normal editor and then submit it to nr just like you'd submit a program to a compiler. Nr formats the input and sends the resultant text to standard output (so you have to redirect it if you don't want to display it on the screen). You invoke nr with:

nr [−switches] files . . . [>stream]

You can list several files—they are just concatenated as the program runs. The command-line switches are optional, and several of them are position sensitive. Table 1, below, summarizes supported switches. They are:

- - —print a list of legal command-line switches.

−*c*—map all control characters, if present, to visible characters before they're printed. This option is partic-

−*c*	don't print (c)ontrol characters
−*d*	print only o(d)d pages
−*e*	print only (e)ven pages
−*m<str>*	append (m)acro: /lib/tmac/<str>.mac
−*n<num>*	(n)umber first page *<num>*
−*o<list>*	print (o)nly pages in list *<list>*
−*p*	suppress bold, underline, overstrike
−*r<str>*	set number (r)eg: −*rx<num>* −*r(xx<num>*
−*s<num>*	(s)top every *<num>* pages
−*t<str>*	set s(t)ring: −*tx<str>* −*t(xx<str>*
−*v*	(v)erbose mode, echo input commands

Table 1: *Summary of command-line arguments*

EVERYTHING THE C PROGRAMMER NEEDS

Eco-C88 C Compiler:

A full, professional C compiler with many ANSI enhancements at an unbelievably low price.

* Prototyping, enum, void data types, plus structure passing & assignment * All operators and data types * Over 200 library functions * cc and mini-make for easy use * 8087 support * ASM or OBJ output * Lint-like tiered error messages * Fast code * CED editor (edit-compile-link from within the editor) * Expanded user's manual * Not copy protected * All for only $59.95!

Eco-C88 Flexi-Graph Graphics Package

Everything you need to write dramatic graphics effects into your Eco-C88 C programs.

* EGA, CGA, and Z100 support * Over 100 graphics functions (many are PLOT-10 compatible) * Most assembler support routines are outside small model code-data * Write thru BIOS (for compatibility) or to memory (for speed) * Graphics function help from CED editor * World, pixel or turtle color graphics modes * 47 standard fill patterns, 17 line dashing patterns, Hershey fonts, plus user defineable fill, dash, and fonts * Supports view areas, rotateable fonts, clipping, arbitrary fill areas, extensive error checking, examples, and user's manual * Only $39.95

Eco-C88 Windowing Library

Use this library to build pop-up windows, help windows, selection menus, special effects—anywhere you need an attention getter.

* CGA and EGA support * Control any program that goes through the BIOS * Use up to 255 windows * No special window commands — use plain old printf( ) to write to a window * Resize and move windows * Custom window titles and borders * Can be used with ANSI device driver * Most window code-data are outside small model * User's manual and examples for only $29.95

Ecosoft Librarian

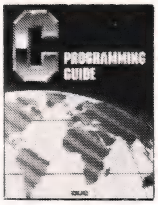
Combine your modules, functions, and subroutines into your own library for easy link commands. Compatible with any standard MSDOS OBJ files * Add, delete, and extract from a library * Get table of contents or index of a library * Combine libraries, control library page size, use switches for combinations, process complex library requests, use wildcards, and do library directives from command files * Complete with user's manual for only $29.95

Developer's Library

Contains the source code for for all library functions, including the transcedentals, memfiles, and those written in assembler. $25.00 with order, $50.00 if ordered later. (Sold only to Eco-C88 owners.)

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

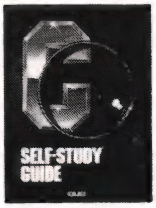
Eco-C88 C compiler requires an IBM PC, XT, or AT (or compatible) with 256K of memory, 2 disk drives and MSDOS 2.1 or later.



C Programming Guide

(Purdum, Que Corp.)

The second edition of the B. Dalton bestseller. Perfect for those just getting started with C. Includes discussion of many X3J11 ANSI Standards Committee recommendations. Many error messages from Eco-C88 give page references to this book. Price is $20.00 plus $2.00 shipping.



C Self-Study Guide

(Purdum, Que Corp.)

Using a question-answer approach, this book is filled with shortcuts, tips, techniques, and traps to avoid when learning C. Price is $17.00 plus $2.00 for shipping.

C Programmer's Library

(Purdum, Leslie, Stegemoller, Que Corp.)

Another B. Dalton bestseller. An intermediate C text for the programmer that wants to get the most from the language. Contains source code for many functions including an ISAM file handler. Price is $22.00 plus $2.00 for shipping.

To order, call or write:

1-800-952-0472 (for orders) or **1-317-255-6476** (tech. info.)

ORDER FORM CLIP & MAIL TO:
Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46220

- ☐ Eco-C88 C Compiler $59.95 ______
- ☐ Eco-C88 Flexi-Graph Graphics Package $39.95 ______
- ☐ Eco-C88 Windowing Library $29.95 ______
- ☐ Ecosoft Librarian $29.95 ______
- ☐ Developer's Library $25.00 ($50.00 if not with order) ______
- ☐ C Programming Guide $20.00 ______
- ☐ C Self-Study Guide $17.00 ______
- ☐ C Programmer's Library $22.00 ______

SHIPPING $4.00
TOTAL (IND. RES. ADD 5% TAX) ______

PAYMENT: ☐ VISA ☐ MC ☐ AE ☐ CHECK
CARD# ______ EXPIR. DATE ______
NAME ______
ADDRESS ______
CITY ______ STATE ______
ZIP ______ PHONE ______

ECOSOFT

(continued from page 96)

ularly useful for debugging escape sequences that are sent directly to the printer. Nonprinting characters are output as *<DD>*, where *DD* is two hex digits.

−d—print only odd-numbered pages. This option is useful if you're sending output to a laser printer and want two-sided output. This command interacts with the *−o* and *−n* switches described later (for example *nr −d −o10−20 file.nr >prn* prints only odd-numbered pages in the range 11 to 19.

−e—print only even-numbered pages.

−m<name>—cause the contents of a macro file to be processed before any of the normal input files are processed. You can think of *−m* as short for /lib/tmac/name.mac. For example, the switch *−ms* causes nr to process the file /lib/tmac/s.mac. If you specify several *−m* options, the files are processed in order from left to right, and all macro files are processed before any normal files are processed.

−n<num>—cause the first page to be numbered *N*—for example, *−n10* causes page numbering to start at 10.

−o<list>—print only those pages in *<list>*. The list can take several forms. The simplest is *−o1,3,5,* which prints only pages 1, 3, and 5. You can specify ranges of pages, as in *−o5−10,* which prints pages 5 to 10 inclusive. The notation *−o−10* means print all pages from the beginning of the document up to and including page 10. Similarly, *−o10−* means print from page 10 to the end of the document. You can combine all these forms, as in *−o−10,12, 15−20,30−*, which prints pages 1 to 10, page 12, pages 15 to 20, and from page 30 to the end of the document. Note that the *−n* option interacts with the *−o* and *−e* options—that is, if you say *−n5,* then saying *−o2* won't work because there is no page 2.

−p—generate plain output (suppress all boldface, underline, and overstrike).

−r<str>—initialize a number register (described later). This option can have two forms:

−rx123
−r(xx123

The first form initializes the single-character number register *x* to *123;* the second initializes the two-character register *xx*. These numbers can be used in the document with *\nx* and *\n(xx* (see later).

−s<num> —stop output every *num*/ pages.This option is useful if you have to hand-feed paper into your printer one page at a time (use *−s1* for this purpose).

−t<str>—initialize a text string macro (works just like *−n* does). The text is available inside the document using the **x* and **(xx* mechanisms, described later. Note that you have to quote the string to get blanks into the text:

nr "*−tx*text with spaces" file

−v (verbose mode)—cause commands to be echoed to standard output just before they're executed but after all the escape sequences (described later) have been expanded. Commands that are part of macro definitions aren't echoed. The name of the input source (a file or macro name) is printed as well. This is a debugging option.

Input to Nr

The command structure and command names nr uses are almost identical to those that nroff uses. There are a few minor differences that I will discuss later. Because I didn't want to create a binary intermediate file, such as the one used by ditroff (device-independent troff), I've added several nonstandard commands to support configuration to various printers. Nonstandard commands are identified as such in the following command descriptions.

One of my original intentions in writing nr was to be able to write documents at home and then upload these to the nroff system at school for final typesetting. Consequently, I tried to make the move as painless as possible. At the macro level, nr is identical to nroff. I've written an implementation of the ms macro package that's in use at UC Berkeley. If your documents are formatted with ms, as are the overwhelming majority of nroff and troff documents, porting to a real Unix system is trivial. The few minor differences between the nr internal commands and the real nroff are well documented and easily translated. I just recently transferred a complete book from nr to the VAX at school and for the most part experienced no difficulties. The main problem I had was with translating macros not in the ms package to the real nroff/troff. Nr is better documented than nroff itself. As a consequence, writing real nroff macros can be difficult. Once you have created the equivalent macros, translation is no problem, of course. The other problems I had were typesetter-related. A typesetter is not a daisywheel printer, and the differences took a few days to figure out.

Nr takes as input a normal ASCII text file that contains intermingled text and formatting commands. Note that nr won't automatically map ASCII to a funny daisywheel—you have to do it yourself. Nr, unlike troff, understands the entire ASCII character set. Some of the characters (such as \) have a special meaning to nr, however, and have to be entered in a special way, discussed later. There's also a provision for printing special non-ASCII characters.

Nr commands take two forms: dot commands and escape sequences. Dot commands all start with a dot in the leftmost column. The dot is followed by a one- or two-letter command name. All of the built-in commands have two-letter names. You can create new commands using nr's macro capability, however, and these can have either one- or two-letter names. There can be any amount of white space (spaces or tabs) between the dot and the first character of the name, which is useful inside a macro if you want to indent the body of an *.if* statement. Because *.if* and *.ie* *(if. . . else)* statements nest, indenting can help a great deal.

Escape sequences, the other sort of command, are text strings that are

embedded in the text itself. They all begin with a backslash (\) but are otherwise dissimilar. You use escape sequences for such tasks as changing fonts on the fly or expanding certain macros. The *\fI* escape sequence, for example, changes the current font to italics and *\fP* puts it back to the previous state. You can put a word into italics with *\fIword\fP*.

Expressions

All the nr commands that take numeric arguments can also take expressions (which are computed as the document is processed) instead of absolute numbers. Several operators are available, shown in Table 2, right. All these operators work just like their C equivalents do except that expression evaluation dosn't terminate when the truth or falsity of an *&&* or *!!* expression is determined. Note that this is a more powerful expression syntax than is supported by the real nroff.

Be careful of strings that follow expressions on the command line. Because white space is legal in an expression, the analyzer just scans the input line until it finds an illegal character. If you say something such as:

```
.vd <up> 1 <down>
```

the < that precedes *down* will be absorbed by the expression parser because < is a legal character in an expression. The problem can be fixed by putting quote marks around the strings:

```
.vd "<up>" 1 "<down>"
```

Most commands treat leading plus or minus signs specially. These signs cause the current value associated with a command to be incremented or decremented by the indicated amount.

For example:

```
.in 10 \" Set indent level to 10
.in +5 \" Increase it to 15
.in −5 \" Decrease it back to 10
```

The \" is a comment; all text that follows it is ignored.

The real nroff supports several unit-of-measurement operators that can be appended onto numbers (inches, picas, points, and so forth). Nr does not support these.

Dot Commands

Nr supports a rich set of dot commands (90 or so). As I mentioned earlier, all commands that take numeric arguments can be passed expressions

Operator	Precedence Level	Meaning
( )	5	used for grouping
−	5	unary minus (as in −5)
!	5	logical NOT
's1's2'	5	compares two strings—evaluates to true (1) if they are equal, to false (0) if they are not
*	4	multiply
/	4	divide
%	4	modulus (MOD)
+	3	addition
−	3	subtraction
<	2	less than
<=	2	less than or equal
>	2	greater than
>=	2	greater than or equal
==	2	equal
!=	2	not equal
&&	1	logical AND
¦¦	1	logical OR

Table 2: *Operators. All operators associate left to right. Higher numbers have higher precedence.*

(continued from page 99)

instead of explicit numbers. The escape sequences on the line are expanded before the expression is evaluated, so you can use number registers and the like in expressions (I'll discuss these in depth in a moment). If a command argument contains any space characters, you must enclose it in double quotes, as in the following:

```
.ds x "several words in a string"
```

Unlike the various Unix shells, the quotes are just for grouping—they do not protect any internal escape sequences (introduced with a \) from expansion. For example:

```
.sp "(\nx + 15)   *   \ny"
```

is treated identically to:

```
.sp (\nx+15)*\ny
```

but is a little easier to read.

All supported dot commands are discussed later. The commands are grouped functionally. Don't be dismayed by their number and complexity. As I mentioned earlier, nr is really a programming language that generates formatted text as output rather than a compiled program. Consequently, you hardly ever have to use the primitive commands themselves; rather, you use subroutines (macros) that are written in terms of the primitive commands. The advantage of a system such as this is that you can redefine the way your text formatter works to suit your convenience.

In all the following descriptions, brackets delimit an optional argument (*[arg]*); in nonliteral arguments, *on* is a string that turns something on and *N* is a number; and angle brackets are used when more than one word is needed to describe an argument (*<left str>*).

Configuration

Nr has several commands that configure it to work with specific printers. Typically these are concentrated in a macro file that is read using the *−m* command-line switch—for example, the switch *−mlaser* tells nr to read the file /lib/tmac/laser.mac before processing other files. The ms macro package I use is configured so that text is displayed properly on the screen, provided that ANSI.SYS is installed—that is, boldface is shown in high intensity, italic is underlined, and so forth.

The configuration commands are:

.bd on off—takes as its argument two strings—one to turn boldface on, the other to turn it off. The maximum length of either string is 80 characters. Use *\x* to send control characters. For example:

Nr is a compiler-like text formatter that can generate output for most printers.

```
.bd \x1b[1m \x1b[0m
```

configures nr for ANSI.SYS. It outputs *ESC[1m* (*0x1b* is an *ESC*) to enable boldface printing. *ESC[0m* turns it off again. If a *.bd* command is never specified, or if a *.bd* is executed with no arguments, then boldface is done by printing every character twice with an intervening backspace (*C<bs>C*). This command is a little different from the one in nroff.

.cm [on]—enable nroff-style copy mode during macro definitions. If an argument is present, nroff copy mode is enabled; otherwise, it's turned off. In normal copy mode only *\"* and *\<CR>* are recognized. In nroff mode the following are recognized:

```
\"  \<cr>  \n  \*  \$  \\  \.  \t  \a
```

Both modes are discussed in greater detail later.

.hd <left str> N <right str>—define horizontal motion. The two strings send the printer cursor left or right by *1/N* spaces. The width of a space is taken from the currently active character-width table (it is 1 in the default monospaced font) and can be changed with a *.ss* command. *N* determines the minimum resolution for the space between characters in proportional-spacing mode. All the widths in the character-width table must be in terms of *N* as well.

As an example, if a space character occupies 12 units of horizontal resolution in a specific font, *N* is 12 and the two strings, when sent to the printer, move the cursor 1/12 of a space width. The character-width tables loaded with the *.df* command (discussed later) contain widths that will all be in terms of these minimum, 1/12 space units. For example, if the character-width table entry for *i* is 6, the character *i* occupies 6/12 of the space occupied by a space. If the entry for *A* is 14, the character *A* takes up 14/12 (1⅙) of the space required for a space character. The default *<left string>* is a single backspace character, the default *<right string>* is a single space character, and the default *N* is 1.

.id on off—send the string specified in *on* to the printer to put it into italics (underline) mode; *off* takes it out. The maximum length of either string is 80 characters. Use *\x<two hex digits>* to send a control character. If no arguments are present or if no *.id* is specified, then underlining is used—nr prints an underscore, a backspace, and then the character for each character.

.od on off—put the printer into overstrike mode (works like *.id* does). A dash is used instead of an underscore in the default situation.

.ss N—change the width of a space in the currently active font to *N*; the default *N* is 1.

.vd <up str> N <down str>—define vertical motion. The *<up str>* string moves the printer cursor up *1/N* lines; the *<down str>* moves it down again.

Font Control and Character Attributes

Several commands are available to change the current font and to load new fonts. Nr handles fonts a little differently from the way nroff does, primarily because most printers han-

dle the various highlight modes somewhat differently from the way that phototypesetters do. All fonts have single-letter names. Five names are reserved by nr:

R—Roman, the default font
I—italics (or underline)
B—boldface
O—overstrike
P—previous

The *R* font is the default font. Initially it is a monospaced (nonproportional) font, but you can replace it with a proportional font by using a *.df* command. You can change the current font with either the *.ft <font name>* command or with an embedded *\f<font name>* escape sequence. For example, you can put the word into italics with *\fIitalics\fP*. Here, *\fI* switches into the italics font (by sending out the string defined with the *.id* command, described earlier), prints the word *italics*, and then switches back to the previously active font with *\fP*.

The *I* (italics) attribute is a little weird in that it's used for both italics and underlining. Typically you can have only one or the other in a document, not both. If you want to have both, you should use nr's italics and then use the line-drawing characters to underline a word when necessary. Note that the real nroff doesn't support an *O* default font. Nr is also different from nroff in that nr treats the *I*, *B*, and *O* fonts as attributes rather than as actual fonts. That is, when you change to font *I*, the current font stays active but nr sends whatever string was defined with the *.id* command out to the printer. This way you can have a bold-italic character by using *\fI\fBword\fP*. Changing to any font other than *I*, *B*, or *O* disables all three attributes.

Font commands are:

.bo [+−]N—put all words on the next *N* input lines into boldface. The default *N* (used when *N* is missing) is 1. Note that this is not an nroff command, though it can be simulated with a macro in nroff. If you want to put an unspecified amount of text into boldface, use:

.bo 1000
<a bunch of text goes here>
.bo 0

.ul [N]—underline (or put into italics) words only on the next *N* input lines. Only alphanumeric characters are underlined; punctuation, spaces, and so on are not.

.cu [+−]N—underline (or italicize) words continuously on the next *N* input lines. All characters are underlined, even spaces and punctuation. For example, <u>This is continuous underlining</u> and <u>this</u> <u>is</u> <u>not</u>.

.os [N]—overstrike the next *N* input lines (works like *.ul* does). If *N* is missing, 1 is used.

.df F <start> <end> <cwidths>—redefine the *R* font (but not the *I*, *O*, or *P* fonts) or add a new font. If no arguments are present, a list of existing fonts is printed to standard output along with the character-width tables.

F is a font name (one character), *<start>* is the name of a macro to invoke when the font is activated in the normal way (with a *\fF* or *.ft F* command), *<end>* is a macro to invoke when you switch out of the font, and *<cwidths>* is the name of a file that holds the character-width table associated with the font. This file must be composed of 256 numbers, with the numbers listed in ASCII order—that is, the first number is the character corresponding to an ASCII *'\0'*, the second number is a Ctrl-B, the 32nd number is the width of the space character, and so on. Numbers must be separated from each other by either white space or new lines.

A sample font-width table is shown in Table 3, below. The 0s on the first line correspond to the characters having numeric values in the range 0 to 31 (all the control characters). A space (ASCII 32) is 12 units wide, an exclamation point (ASCII 33) uses 6 units, a double-quote mark (ASCII 34) uses 8 units, and so on. If numbers are missing from the end of the list, 1 is assumed. A *unit* here must also be defined in the *.hd* command described earlier. If no font-width file is specified to *.df*, a table is created and all entries in it are set to 1. (This is the default for a monospaced font.)

.ft F—change to font *F* at the beginning of the next input text line. You can also embed font changes with a *\fF* escape sequence. Note that, if font *F* doesn't exist, the error won't be flagged until the output routines try to process the font change request.

Text Filling, Adjusting, and Centering

Nr generally fills lines—that is, it collects words from input (a word is any space-delimited collection of characters) until it has collected an entire output line, and then it outputs all the words on a single line. For example:

This
is several
words.

will be collected and printed as:

This is several words.

If hyphenation is enabled, it will read one word too many and then try

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	6	8	12	10	16	14	6								
6	6	10	10	6	8	6	8								
10	10	10	10	10	10	10	10								
10	10	6	6	10	10	10	10								
16	14	12	14	14	12	12	14								
14	6	10	14	12	16	14	14								
12	14	14	10	12	14	12	16								
14	14	12	6	8	6	10	12								
10	10	10	10	10	10	8	10								
10	6	6	10	6	16	10	10								
10	10	8	8	8	10	10	14								
10	10	10	6	6	6	10	0								

Table 3: *A font-width file*

(continued from page 101)

to hyphenate the last one. If nr can insert a hyphen to squeeze more characters onto the current line, it will do so. You can also adjust the text in several other ways. The most common is to insert white space between words in order to get the rightmost characters to line up (with the words spread as evenly as possible on the line).

You can force a line break (in which the contents of the fill buffer are printed even if there aren't enough words to fill the line) in several ways. The *.br* command always causes a break and leaves the cursor at the beginning of the next output line. In addition, several other commands—*.bp*, *.br*, *.ce*, *.fi*, *.in*, *.nf*, *.sp*, and *.ti*—cause breaks as a side effect of their operation. If you don't want a line to break when one of these is executed, replace the dot that's usually used to introduce a command with the *nobreak* command character (the default is a backquote [´]). For example, the *.sp N* command usually causes a break and then prints *N* blank lines. The *'sp N* command, however, prints the blank lines without flushing the fill buffer first. You can change the default no-break character with a *.c2* command.

Commands for controlling filling and margins are:

.ad [C]—turn on margin adjusting. Adjustment modes (values of *C*) are:
- *b*—adjust both margins.
- *n*—same as *b*.
- *l*—adjust only the left margin, leaving a ragged-right edge, as in a hand-typed document.
- *r*—adjust only the right margin, leaving a ragged-left edge. God knows what this mode is good for, but nroff supports it.
- *c*—center each output line on the page.

If *C* is missing then the most recently active adjustment mode is used.

.br (break)—print all the words in the current fill buffer even if there aren't enough words to fill the output line, then go to the next output line.

.ce [N]—center the next *N* input lines without filling. Default *N* is 1. This command causes a break.

.fi—enable line filling. The default is filling off, so a *.fi* command must be specified at the top of the input. This is usually done automatically by a macro file such as ms. This command causes a break.

.na—turn off adjusting. Turn it back on with a *.ad*.

.nf—disable line filling, flushing the buffer first. This command causes a break.

Page Control

Nr has several commands for page control:

.bp [[+−] N]—begin page *N*. If *N* is absent, use the current page number plus 1. Note that *N* is the number of the new page, not the current one, so a footer on the current page will reflect the old number. If *N* has a leading plus or minus sign, the current page number is modified by the indicated amount. This command causes a break.

.ne N—need *N* lines. If there aren't that many lines on the current page, then force a new page. The *.ne* command actually looks at the distance from the current position on the output page to the next output line trap, discussed later in the Macros, Strings, Diversions, and Traps section. If this distance is less than *N*, nr skips forward to the trap. The assumption here is that the trap will be an end-of-page trap.

.pl [+−]N—set page length to *N* lines.

.po [+−]N—set page offset to *N* spaces. The page offset is a specified number of space characters that are printed to the left of every output line—that is, *.po* defines the width of the left margin.

Changing Special Characters

Certain characters are special to nr. These are:

.(command character)—introduces

C Programmers! High-Speed Database tames complex C applications

"db_VISTA™ lets you easily build complex databases with many interconnected record types. . ."
Dave Schmitt, President, Lattice, Inc.

High-Speed data retrieval and access. . . just two benefits of using RAIMA's network model DBMS, db_VISTA. Combine these design benefits with those of C—speed, portability, efficiency, and you begin to understand db_VISTA's real measure. . . performance.

Independent Benchmark proves High-Speed model 2.76 times faster

An independent developer benchmarked db_VISTA against a leading competitor. Eleven key retrieval tests were executed with sequentially and randomly created key files.

*Result of 11 Key Retrieval Tests

db_VISTA	:671.24 seconds
Leading Competitor	:1,856.43 seconds

db_VISTA's high-speed network database model lets you precisely define relationships to minimize redundant data. Only those functions necessary for operation are incorporated into the run time program.

Portable DBMS Applications with db_VISTA

For maximum application portability, every line of db_VISTA's code is written in C and complete source code is available. db_VISTA operates on most popular computers with several operating systems supported. So whether you write applications for micros, minis, or mainframes. . .db_VISTA is for you.

How db_VISTA works. . .

Design your database and compile your schema file with the database definition language processor. Develop application programs, making calls to db_VISTA's C functions. Edit and review your database using the Interactive Database Access utility. Compile and link your C program with the db_VISTA runtime library, and your application is ready to run.

Multi-user and LAN capability

Information often needs to be shared. db_VISTA has multi-user capability and supports simultaneous users in either multi-tasking or local area networking environments, allowing the same C applications to run under UNIX, MS-DOS, and VAX VMS.

db_QUERY™ lets you ask more of your database

db_QUERY is a linkable, royalty-free, SQL-based ad hoc query and report writing facility. It provides a user-friendly relational view of a network-model database. Use it directly or design an interface for the inexperienced to generate powerful queries and reports.

Royalty-Free Run-Time

Whether you're developing applications for a few customers, or for thousands, the price of db_VISTA or db_QUERY is the same. If you are currently paying royalties for a competitor's database, consider switching to db_VISTA and say goodbye to royalties.

FREE Technical Support For 60 days

Raima's software includes free telephone support and software updates for 60 days. Technical support personnel are available to answer questions about our software or yours.

30-Day Money-Back Guarantee

Try db_VISTA for 30 days and if not fully satisfied, return it for a full refund.

Order Schedule

	db_VISTA	db_QUERY
☐ Single-user	$ 195	$ 195
☐ Single-user w/Source	$ 495	$ 495
☐ Multi-user	$ 495	$ 495
☐ Multi-user w/Source	$ 990	$ 990
☐ VAX Multi-user	$ 990	$ 990
☐ VAX Multi-user w/Source	$1980	$1980

Not Copy Protected

Call Toll-Free Today!

Order Line 1-800-327-2462
Information Line . 1-206-828-4636

Read what others say. . .

"If you are looking for a sophisticated C programmer's database, db_VISTA is it. In either a single or multi-user environment, db_VISTA lets you easily build complex databases with many interconnected record types. The multi-user implementation handles data efficiently with a LAN, and Raima's customer support and documentation are excellent. Source code availability and a royalty-free run-time is a big plus."
Dave Schmitt, President Lattice, Inc.

"My team has developed a sophisticated PC-based electronic mail application for resale to HP customers. db_VISTA has proved to be an all-round high performer in terms of fast execution, flexibility and portability, and has undoubtedly saved us much time and development effort."
John Adelus, Hewlett-Packard Ltd. Office Productivity Division

"On the whole, I have found db_VISTA easy to use, very fast with a key find, and powerful enough for any DBMS use I can imagine on a microcomputer."
Michael Wilson, Computer Language

db_VISTA Version 2.2

Database Record and File Sizes
- Maximum record length limited only by accessible RAM
- Maximum records per file is 16,777,215
- No limit on number of records or set types
- Maximum file size limited only by available disk storage
- Maximum of 255 index and data files

Keys and Sets
- Key length maximum 246 bytes
- No limit on maximum number of key fields per record—any or all fields may be keys with the option of making each key unique or duplicate
- No limit on maximum number of fields per record, sets per database, or sort fields per set
- No limit on maximum number of member record types per set

Operating System & Compiler Support
- Operating systems MS-DOS, PC-DOS, UNIX, XENIX, SCO XENIX, UNOS, ULTRIX, VMS
- C compilers: Lattice, Microsoft, DeSmet, Aztec, Computer Innovations, XENIX and UNIX

Features
- Multi-user support allows flexibility to run on local area networks
- File structure is based on the B-tree indexing method and the network database model
- Run-time size, variable—will run in as little as 64K, recommended RAM size is 256K
- Transaction processing assures multi-user database consistency
- File locking support provides read and write locks on shared databases
- SQL-based db_QUERY is linkable
- File transfer utilities included for ASCII, dBASE optional

Utilities
- Database definition language processor
- Interactive database access utility
- Database consistency check utility
- Database initialization utility
- Multi-user file locks clear utility
- Key file build utility
- Data field alignment check utility
- Database dictionary print utility
- Key file dump utility
- ASCII file import and export utility

*The benchmark procedure was adapted from "Benchmarking Database Systems: A Systematic Approach" by Bitton, DeWitt and Turbyfill, December 1983.

RAIMA™ CORPORATION

High-Speed Programming Tools, Designed for Portability

3055-112th Avenue N.E.
Bellevue, WA 98004 USA
(206) 828-4636 Telex: 9103330300

Order Toll-Free
1 (800) 327-2462

Circle no. 206 on reader service card.

(continued from page 102)

dot commands.

(nobreak character)—also introduces commands, but a line break is not done if that command usually forces a break.

\ (escape character)—introduces an escape sequence.

You can change these characters with the following:

.c2 [C]—change no-break character to *C*. If *C* is missing, use a backquote (′).

*.cc [*C*]*—change command character to *C*. If *C* is missing, use a period (.).

.ec c—change escape character to *C*. If *C* is missing, use a backslash (\).

.eo—disable the escape mechanism entirely (change the escape character to nothing). You can restore it again with a *.ec* command.

Spacing, Line Length, and Indenting

You can use the commands listed in this section to change line spacing, the current indent level, and so forth. If you use a leading plus or minus sign in a numeric argument, the current value is modified by the indicated amount; otherwise, the current value is changed to the indicated value.

.in [+−]—change the indent level to *N*. This indent is in addition to the left margin, which is set up with the *.po* command, described earlier. If you use both *.po* and *.in*, then the left margin is the sum of the values given to the two commands. Generally the page offset remains constant throughout a document, and the indent is changed with *.in*. This command causes a break.

.ll [+−]N—change line length to *N* spaces. The line length determines how many words are collected when line filling is enabled.

.ls [+−]N—change line spacing to *N* lines; 1 is single spacing, 2 is double spacing, and so on.

.ns—inhibit the printing of blank lines (no-space mode)—that is, no blank lines will be printed until some text is encountered, a *.bp N* is executed (the *N* is required), or a *.rs* is executed. This command is useful in the top-of-page macro.

.rs—restore blank line printing when it has been turned off with a previous *.ns* command.

.sp [N]—space down *N* lines (print *N* blank lines). *N* can be negative if your printer supports reverse line feeds and a previous *.vd* command was executed. This command causes a break. Note that a blank input line is treated identically to a *.sp 1* (it forces a break and prints a blank line under the flushed text).

.ti N—set the temporary indent to *N* spaces—that is, only the next output line will be indented by the indicated amount. This command is useful for the first line of a paragraph. The indent level for this line will be the sum of the indents specified in the *.po* and *.ti* commands—that is, the *.in* command isn't used in the calculation. To indent relative to the current indent level, use a leading plus or minus sign. For example, *.ti +5* causes the next line to be indented five spaces further than the current indent level, as specified with previous *.in* or *.po* commands. The *.ti* command causes a break.

Macros, Strings, Diversions, and Traps

Macros are the heart of nr. Without them the word processor would be so difficult to use that it wouldn't be worth the trouble. Macros are collections of text. When you define a macro, the text is saved by nr, and when you expand a macro, the text is used for input. The mechanism is identical to the *#define* mechanism in C. A macro name can be any length, though for nroff compatibility I'd suggest limiting yourself to one- or two-character names. Macros used in traps (discussed shortly) must have one- or two-character names, however. Macro names are case sensitive. You cannot define a macro that has the same name as a built-in dot command (if you do, the macro will just be ignored).

A macro is defined with a *.de <name>* command and is expanded as if it were a dot command whenever you precede its name with a dot in the first column. A macro can take up to nine arguments (accessible within the macro using *\$1*, *\$2*, and so forth). For example a macro defined with:

```
.de xx
arg 1 <\$1>
arg 2 <\$2>
arg 3 <\$3>
...
```

is invoked with:

```
.xx "this is one argument" doo wha
```

and will print:

```
arg 1 <this is one argument>
arg 2 <doo>
arg 3 <wha>
```

Macros can also call other macros (though recursion is not permitted). In practice they are used in the same way as subroutines are. They let you take the nr primitives described here and do something useful with them.

There are two flavors of macros: true macros and strings. A true macro is intended to hold a collection of commands and text; a string is intended to hold text that is expanded into a line. In practice, the only difference is that the last line in a macro is terminated with a carriage return whereas the last line of a string is not. Strings are defined with a *.ds* or *.as* command. They are expanded using the **x* or **(xx* escape sequences. The first syntax is for one-character names, and the second is for two-character names. Strings may contain escape sequences. Note that they are defined in normal mode, however (not in copy mode as the real nroff does it). This means that you need to use double backslashes to get an escape sequence into a string. For example, if you want to define a string called *#d* that prints the word *#define* in boldface, you could use:

```
.ds #d \\fB#define\\fP
```

The string could be used later by embedding a **(#d* into the text where you wanted the word to appear.

A diversion is a macro that's used to

delay printing temporarily. This way you can collect footnotes or a table of contents in a diversion and then print the diversion out at the end of the document. The *.di xx* command causes output to be sent to the macro called *xx* rather than to the output stream. A *.di* without arguments will stop the redirection and restore the previous output stream. Diversion nesting is permitted—you can redirect to a diversion from within a diversion.

Macros and diversions are both created in copy mode, a crippled input mode in which only two escape sequences are recognized (\" and \<CR>). Copy mode is described in greater depth later. Nr supports two copy modes—the one just described and an nroff-compatible copy mode that is a little less restrictive. Small macros are stored internally, in RAM. If the macro gets too large (greater than 256 characters), it is stored on disk, however. The file names all take the form xxxx.mac, where *xxxx* is four hex digits. The string defined in the *TMP* environment (created by COMMAND.COM with a *set* command and by the shell with a *setenv* command) is appended to the front of the file name, so you can use something such as:

set TMP d:/tmp/

to put macro files onto a RAM disk. The trailing / is necessary here. Macro files are all deleted when nr terminates.

One of the more useful features of nr is a trap. A trap is way to tell nr to expand a macro automatically when a specified event occurs. For example, you can set a trap to expand a macro at the top or bottom of every page. You can spring a trap after a specified number of input lines have been read or after a specified number of lines have been put into a diversion. There's also a special trap that's sprung once, after the entire document has been printed.

.de name [xx]—define a macro and give it the indicated *name.* All lines between the *.de* command and the first line that begins with . . (or with *.xx*, where *xx* is the second argument to *.de*) are added to the macro. If a macro with the indicated name exists, it is destroyed. If both arguments are missing, all currently defined macros are printed (like *.pm* in real nroff does, except the contents of the macro are printed, too).

.am name [xx]—append text to an existing macro. It works like *.de* does but doesn't overwrite the existing macro.

.ds name text—define a string called *name*, and put the indicated *text* into it. If the string already exists, it is deleted.

.as name—append text to the end of an existing string. It works like *.ds* does except that it doesn't overwrite the existing string.

.di [name]—divert output to the named macro. The diversion is terminated by a *.di* or *.da* command that has no argument. Diversions can be nested. Normal text processing occurs in a diversion except that the page offset isn't done. If a macro having the indicated name already exists, it is destroyed.

.da [name]—divert text to the named macro, appending to its end rather than overwriting it. Stop appending when a *.da* or *.di* without an argument is encountered.

.rm name—remove the named macro or string. If the macro is on the disk, the file is deleted.

.em name—use the named macro as the end macro. This macro will be executed once, after all output has been processed. You can't give arguments to the end macro.

.wh N [name]—set an output line trap. The named macro is executed automatically, immediately after printing line *N* on every page. If *N* is 0, the trap is sprung at the top of a page (above line 1). If the *name* is absent, the trap at line *N* is removed. If *N* is negative, then the trap is set relative to the bottom of the page. (The location is determined by looking at the page length [as set with *.pl*] that was in effect when the *.wh* was executed.) The macro replaces any previously installed macro for that trap position; macros do not shadow one another as in the real nroff.

.ch name [[+ −]N]—change output line trap position for the named macro to line *N*. Any existing trap at that position is destroyed (nroff will shadow the earlier trap, not destroy it). If *N* is absent, the trap is removed.

.dt [+ −]N name—set a diversion trap. The named macro is executed after *N* lines have been written into the current diversion. Only one diversion trap may be active.

.it [+ −]N name—set an input line trap. Execute the named macro after *N* lines of input have been read. Only one input line trap may be active. A *.it* destroys a previous trap if one exists.

Environments

In a real programming language you can copy things into local variables when you need to save them. Nr, however, only supports global variables, and this can present a problem. A solution of sorts is the environment mechanism. An environment is a stack. When you save an environment, various parameters that control how nr works are pushed onto the stack. You can then change those parameters at will. The old environment can be be popped from the stack at a later date, overwriting any changes that were made after the previous push. The saved parameters are listed in Table 4, page 108.

The *.ev [N]* command pushes various commonly used variables onto an environment stack. Nroff supports several environments, and nr supports only one. If an argument is present, the current environment is pushed. If no argument is present, a previously saved environment is popped from the stack. The stack can hold up to five environments. An error message is printed if you try to push more than five.

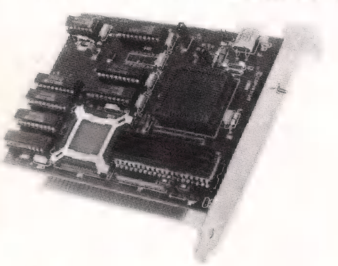
Number Registers

Number registers are nr's global variables. They are used to hold numeric quantities. You create number registers with a *.nr* command and expand them into the text with *\nx* or *\n(xx* escape sequences. The first syntax is for one-character names, and the second is for two-character names. The string *\nx*, when found in the input, is replaced by a string representing the

MICROWAY MEANS 8087 PERFORMANCE

FastCACHE-286™

Runs the 80286 at 8.5 or 11 MHz and the 80287 at 5, 6 or 11 MHz. Includes 8 kbytes of 55ns CACHE. Works with more PCs than any other accelerator, including Leading Edge Model D, Compaq, and Turbo motherboards. Includes 8088 Reboot Switch, DCache and Diagnostics........................ **From $449**



LOTUS/INTEL EMS SPECIFICATION BOARDS

MegaPage™ The only EMS board which comes populated with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache and EMS drivers. For the IBM PC, XT and compatibles...**$549**

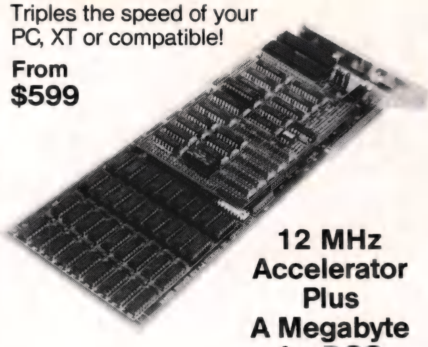
MegaPage with ØK............... **$149**

MegaPage AT/ECC™ EMS card for the PC AT and compatibles includes Error Correction Circuitry. With ECC, 11 RAM chips cover 256K so the user never encounters RAM errors. Sold populated with 1 megabyte CMOS . . . **$699** or with 3 megabytes CMOS cool running low power drain RAM . . . **$1295**. Optional serial/parallel daughterboard....... **$95**

NUMBER SMASHER/ECM™

Triples the speed of your PC, XT or compatible!

From $599



12 MHz Accelerator Plus A Megabyte for DOS

***PC Magazine* "Editor's Choice"**

DATA ACQUISITION and REAL TIME TOOLS

DAL™ – "Data Acquisition Language."

Unkelscope™ – A real time data acquisition, control and process software pkg.

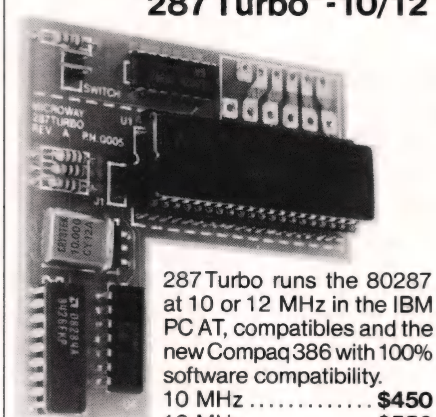
87 FFT and 87 FFT-2

TransView Menu driven FFT Spectrum/transfer analyzer.................... **$250**

RTOS – REAL TIME OPERATING SYSTEM A multi-user, multi-tasking real time operating system. Includes a configured version of Intel's iRMX-86, LINK-86, LOC-86, LIB-86, OH-86 and the MicroWay 87DEBUG. Runs on the IBM-PC, XT, PC-AT and COMPAQ................... **$600**

INTEL COMPILERS Available for RTOS FORTRAN-86, PASCAL-86, PL/M-86.

287 Turbo™-10/12



287 Turbo runs the 80287 at 10 or 12 MHz in the IBM PC AT, compatibles and the new Compaq 386 with 100% software compatibility.

10 MHz **$450**
12 MHz **$550**

***PC Magazine* "Editor's Choice"**

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and installation instructions.

8087 5 MHz................. **$114**
For the IBM PC, XT and compatibles

8087-2 8 MHz............... **$149**
For Wang, AT&T, DeskPro, NEC, Leading Edge

80287-3 5 MHz............. **$179**
For the IBM PC AT and 286 compatibles

80287-6 6 MHz............. **$229**
For 8 MHz AT compatibles

80287-8 8 MHz............. **$259**
For the 8 MHz 80286 accelerator cards

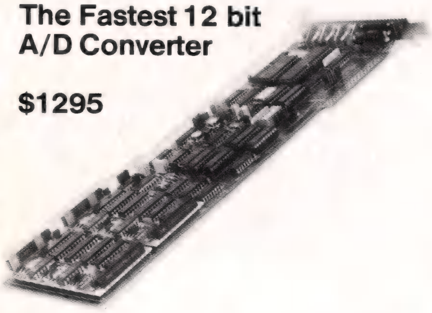
80287-10 10 MHz......... **$395**
For the Compaq 386

Call for prices on V20, V30, 64K, 128K and 256K RAM

A2D-160™

The Fastest 12 bit A/D Converter

$1295



160,000 Samples per second
Pseudo Random Noise Generator/DAC
Optional signal conditioners
AFM-50 Programmable Low Pass Filter Module..................... **$225**

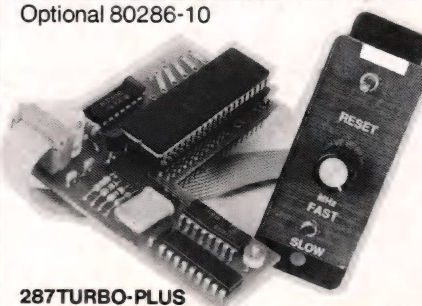
8087 SOFTWARE

IBM BASIC COMPILER	$465
MICROSOFT QUICK BASIC	$79
87BASIC COMPILER PATCH	$150
IBM MACRO ASSEMBLER	$155
MS MACRO ASSEMBLER	$99
87MACRO/DEBUG	$200
MICROSOFT FORTRAN	$209
RM FORTRAN	$399
LAHEY FORTRAN F77L	$477
MS or LATTICE C	CALL
STSC APL★PLUS/PC	$450
STSC STATGRAPHICS	$675
SPSS/PC+	$675
87SFL Scientific Functions	$250
PHOENIX PRODUCTS	CALL
FASTBREAK for 1-2-3 V.1A	$79
HOTLINK for 1-2-3 V.1A	$99

287TURBO-PLUS™

Speeds up your AT

Adjustable 80286 Clock 6-12 MHz
10 MHz 80287 Clock
Plus Full Hardware Reset......... **$149**
Optional 80286-10



287TURBO-PLUS
With 80287 10 MHz.................... **$549**
With 80287 12 MHz.................... **$629**

CALL (617) 746-7341 FOR OUR COMPLETE CATALOG

MicroWay P.O. Box 79, Kingston, Mass. 02364 USA (617) 746-7341

The World Leader in 8087 Support!

MicroWay Europe
32 High Street
Kingston-Upon-Thames
Surrey England KT1 1HL
Telephone: 01-541-5466

Circle **no. 300** on reader service card.

(continued from page 105)

contents of the indicated number register.

There are preincrement and decrement syntaxes, too: \n+*x*, \n+(*xx*, \n−*x*, and \n−(*xx*. With these syntaxes the number register is incremented (or decremented) by a predetermined amount before the escape sequence is interpolated. Nonexistent number registers expand to *0*. You can use number registers both in commands and embedded in the text.

Several number registers are created and maintained by nr itself (see Table 5, below). These hold such things as the current page number.

You can use number registers to do things such as keep track of the current footnote number. For example, *.nr fn 0* creates a number register called *fn* that will hold the footnote number. You can access this register with \n(*fn*, but if you use \n+(*fn*, then the register will be incremented automatically before it's expanded. This process can in turn be hidden in a string—*.ds * \u\\n+(fn\d*. Here the \u and \d send the cursor up and down half a line. You need two backslashes to prevent nr from expanding the number register at definition time. You can now expand the string with a ** in the text, thereby both printing and incrementing the current footnote number. The number register is incremented before it's expanded.

Number registers can be expanded into the text in several formats. That is, the number is just a number, but it can be expanded as an Arabic number (with optional zero fill), as an uppercase or lowercase Roman numeral, in outline format (*a, b, c, z . . . aa, ab . . . az*), or in English words (one thousand, two hundred fifty-seven).

.nr name [+−]N [[−]M]—create or modify number register *name* by *N*. For example, *.nr x 10* creates a number register called *x* and initializes it to 10, and *.nr x +5* increases the contents of *x* to 15. *M* is the increment amount (the default is 1 if *M* is absent). When the number register is accessed using the \n+*x* or \n+(*xx* syntax, then *M* is added to the register before it is expanded. *M* may be negative. Unlike nroff, *.nr*, with no arguments, prints a list of all currently defined number registers and their contents.

.rr name—remove the named number register.

.af name mode—alter the expansion format of the named number register to the indicated mode. Default is Arabic. Legal values of *mode* are shown in Table 6, page 109. The leading 0s in the Arabic formats (as in the second and third lines of Table 6) determine the field width of the number.

In my next column, I'll conclude this user's manual by describing tabulation, control flow, hyphenation, line numbering, and more.

Availability

The February, March, and April 1987 C Chests have been combined to cre-

The following parameters are saved:

- the unprinted contents of the fill buffer (the buffer is cleared after its contents are stored)
- the input line trap (it's cleared after being saved)
- the count associated with the *.cu*, *.ul*, *.bo*, or *.os*—all these are set to zero after being saved
- the adjustment mode (as set with *.ad*)
- the current font (as set with *\f* or *.ft*)
- the command character (as set with *.cc*)
- the escape character (as set with *.ec*)
- the current no-break character (as set with .c2)
- the fill status (line filling enabled [.fi] or disabled [*.nf*])
- the indent level (as set with *.in*)
- the page offset (as set with *.po*)
- the line spacing (as set with *.ls*)
- the line-numbering values (as set with *.nm*)
- the margin characters (*.mc* and *.lm*)
- the tab stops and the tab and leader expansion characters
- the line length (*.ll*)
- the temporary indent (*.ti*)
- the three-part title length (*.tl*)

***Table 4:** The contents of an environment*

%	current output page number
dl	width of (maximum line length of any line in) the most recently completed diversion
dn	height of (numbers of lines in) most recently completed diversion
dy	day when execution started (1−31)
h	hour when execution started (1−24)
hp	current horizontal place on input line
ln	current line number used by the *.nm* command for line numbering.
m	minute when execution started (1−59)
nl	current output line number (used by *.nm*)
mo	month when execution started (1−12)
s	second when execution started (1−59)
wd	day of the week when execution started (1−7, 1 is Sunday)
yr	year (for example, 1987)
.$	number of arguments to the current macro
.c	number of lines read from current input file.
.d	vertical place in current diversion (distance from line 1).
.f	currently active font (can be stored and then passed to *.ft* later on).
.i	current indent column (as set with a *.in*)
.l	current line length (as set with a *.ll*)
.n	length of the text part of previous output line
.o	current page offset (as set with *.po*)
.p	current page length (as set with *.pl*)
.t	distance to next trap (in lines) (very large if there's no trap)
.u	1 if in fill mode, 0 otherwise
.v	current line spacing (as set with *.ls*)

***Table 5:** Predefined number registers*

ate *Nr: An Nroff-Like Text Processor for MS-DOS*. This reprint is available with a source-code disk for $29.95. Send prepaid orders to M&T Books, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600, extension 216. Please add $2.25 for shipping and handling ($5 for foreign orders).

DDJ

(Listings begin on page 48.)

Mode	Register Expands As
1	1, 2, 3, 4, . . .
01	01, 02, 03, 04, . . .
001	001, 002, 003, 004, . . .
i	i, ii, iii, iv, v, vi, vii, . . .
I	I, II, III, IV, V, VI, VII, . . .
a	a, b, c, . . . z, aa, ab, ac . . . az, ba, bb, . . .
A	A, B, C, . . . Z, AA, AB, AC . . . AZ, BA, BB, . . .
e	one, two, three, four, five, six, . . .
E	One, Two, Three, Four, Five, Six, . . .

Table 6: *Number register output formats*

Flotsam and Jetsam

Definitions, Declarations, and Casts

Kernighan and Ritchie, for reasons unknown to myself, use the terms *declaration* and *definition* in a special way. Unfortunately, the way they use these words is the inverse of the way in which every other programmer thinks of them. A declaration is an announcement (at least according to *Webster's*). Consequently K & R use the word *declaration* to mean that you are announcing the presence of a variable to the compiler. You aren't allocating space for that variable; you're just announcing its presence somewhere in some module in your program. The linker will find the actual variable when the modules are linked. An *extern* statement is used to declare a variable in K & R's sense of the word.

On the other hand, *Webster's* says that to define an object is to "fix or mark the limits" of that object, to allocate space for the object. So a variable definition in C is what actually allocates space for a variable. This usage is backward from the normal usage, thus the confusion. A declaration is always implicit in a definition—when you allocate space for an object (define it), you also tell the compiler that the object exists somewhere (here).

The declaration/definition conundrum can cause problems. A particularly nasty one is brought about by implicit declarations of subroutines. If you use a subroutine that hasn't been previously declared (with either an *extern* statement or a real definition), the compiler assumes that the subroutine returns an *int*. The problem arises when you then use the cast operator in conjunction with the implicit declaration.

A cast operator temporarily changes the type of a specific object. It is formed by writing a variable declaration of the required type, surrounding the declaration with parentheses, and then removing the name and semicolon. For example, you'd declare a character pointer with:

```
char   *Dostoevski ;
```

You change the declaration to a cast by surrounding the foregoing with parentheses:

```
(char   *Dostoevski ;)
```

and removing the name and semicolon:

```
(char   *)
```

You can now convert an object to a character pointer by preceding its use with the cast.

An example: you've defined an *int*-size variable called *baton* and want to pass it to a subroutine called *runner( )*, which expects a *double*-size argument. You can force an *int*-to-*double* type conversion with a cast:

```
runner( (double) baton );
```

The definition/declaration problem arises when you try to use a cast to change the type of an object that was implicitly declared as type *int*. For example, the following will not work as expected in the 8086 medium or large models:

```
struct building *tourist ;
tourist = (struct building *)
        malloc( sizeof(struct building) );
```

You had intended to convert the character pointer returned from *malloc( )* into a building pointer. The compiler doesn't know that *malloc( )* returns a character pointer, however. It assumes that *malloc( )* returns an *int* because there's no preceding *extern* statement. Pointers and *ints* are different sizes in the 8086 medium or large models, however. (An *int* is probably 16 bits wide, and a pointer is probably 32 bits wide.) Because the compiler thinks that *malloc( )* returns an *int*, it truncates the 32-bit pointer down to 16 bits—the size of an *int*. Only now will it look at the cast operator, converting the *int* back to a pointer. Unfortunately, the precision that you lost when the variable was truncated is still lost. That is, the upper 16 bits of the pointer are lost forever, converted to 0s.

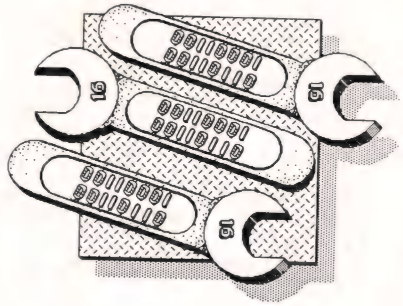
You can fix the problem by telling the compiler that *malloc( )* indeed returns a pointer of some sort. Use either:

```
extern char   *malloc( );
tourist = (struct building *)
                        malloc( . . . );
```

or:

```
extern struct building *malloc( );
tourist = malloc( . . . );
```

In the first example, you're converting a character pointer to a building pointer. As both of these pointers are the same width, no precision is lost.



80386 Resources

Attendees of the November 1986 Comdex in Las Vegas found themselves deluged by Intel 80386 hype and hysteria from vendors and press alike. I'll be discussing this interesting new supermicro at length in these pages after Santa brings me a 80386-based machine or accelerator board to play with; in the meantime, here are some helpful sources of information:

80386 Programmer's Reference Manual. About 350 pages. Intel order number 230985-001.

This manual covers architecture, memory management, memory protection, multitasking, input/output, exception and interrupt handling, debugging support, virtual 8086 mode, and mixing 16-bit and 32-bit code, and it has a full reference section on the individual instructions. It's a must-have for would-be 80386 programmers.

80386 Hardware Reference Manual. Intel order number 231732-001.

This book covers internal architecture and pipelining, local bus interface, coprocessor interface, and memory cache. It's for hardware knowledgeable types only.

Introduction to the 80386. Intel order number 231252-001.

A nice readable overview of the 80386 in its native 32-bit processing mode and its support for paging, memory protection, and multitasking. It also includes a discussion of upward compatibility from 8086, 286 protected mode, and the virtual 86 mode.

by Ray Duncan

80386: A Collection of Article Reprints. 60 pages. Intel order number 231737-001.

A compilation of recent feature articles from *Electronic Design*, *IEEE Micro*, *Computer Systems*, and *Tech Notes*.

The 80386: A High Performance Workstation Microprocessor. Intel order number 231776-001.

An evaluation of the throughput of the 80386 and comparisons with other popular processors. It includes the C source code for the Dhrystone and Whetstone benchmarks.

80386 High Performance 32-Bit Microprocessor with Integrated Memory Management. Product data sheet dated April 1986. 131 pages. Intel order number 231630-002.

A very terse summary of the hardware reference and programmer's reference mentioned earlier.

You can order all the above from Intel Literature Sales, P.O. Box 58130, Santa Clara, CA 95052-8130; (800) 548-4725. Intel's telephone order service is courteous, and delivery is prompt. The Intel publication catalog, order number 210620-010, is free for the asking.

80386 Un-Resources

Murray, William H., III, and Pappas, Chris H. *80386/80286 Assembly Language Programming*. Berkeley, Calif.: Osborne/McGraw-Hill, 1986. 548 pages with index. ISBN 0-07-881217-8.

This is the 80386 reference book not to buy; it is a sad example of a publisher's unscrupulous attempt to cash in on a new technology. Murray's book is essentially about 8086 programming with a few nods to the additional instructions and protected mode of the 80286, and it makes only token references to the 80386. The few program fragments that illustrate the 80386's 32-bit instructions would never run if assembled in current environments because they don't include the 32-bit override byte. Some of the more interesting features of the 386, such as caching, pipelined instruction execution, segments up to 4 gigabytes in length, and bit instructions, are not covered at all.

Assembly-Language Resources

The November/December 1986 issue of *Programmer's Journal* contains two articles that 16-Bit Toolbox readers will find especially useful. M. Steven Baker has contributed an explanation of Terminate and Stay Resident utilities that includes discussion of the In-DOS flag (*int 21h*, function *34h*) and the Multiplex Interrupt (*int 2fh*). George Defenbaugh has written an article on "Parents, Children, Redirection, and Piping" that discusses the MS-DOS *DUP* and *CDUP* functions (*int 21h*, functions *45h* and *46h*).

The Byte Information Exchange (BIX) has an exceptionally active and useful conference called MS-DOS Secrets. This conference already contains nearly a thousand messages about undocumented MS-DOS interrupts, TSR techniques, MS-DOS bugs and work-arounds, and the like. If you are a serious MS-DOS programmer, you will find the cost of a BIX account more than justified by this conference alone.

William Claff was kind enough to send me copies of the first eight issues of his monthly newsletter, *PC Tech Report*. These issues cover such topics as the *ASSUME* and *GROUP* directives, making .EXE files resident, device driver templates, 8087 programming, and a complete critical error

SAS Institute Inc. Announces

Lattice C Compilers for Your IBM Mainframe

Two years ago...

SAS Institute launched an effort to develop a subset of the SAS® Software System for the IBM Personal Computer. After careful study, we agreed that C was the programming language of choice. And that the Lattice® C compiler offered the quality, speed, and efficiency we needed.

One year ago...

Development had progressed so well that we expanded our efforts to include the entire SAS System on a PC, written in C. And to insure that the language, syntax, and commands would be identical across all operating systems, we decided that all future versions of the SAS System—regardless of hardware—would be derived from the same source code written in C. That meant that we needed a C compiler for IBM 370 mainframes. And it had to be good, since all our software products would depend on it.

So we approached Lattice, Inc. and asked if we could implement a version of the Lattice C compiler for IBM mainframes. With Lattice, Inc.'s agreement, development began and progressed rapidly.

Today...

Our efforts are complete—we have a first-rate IBM 370 C compiler. And we are pleased to offer this development tool to you. Now you can write in a single language that is source code compatible with your IBM mainframe and your IBM PC. We have faithfully implemented not only the language, but also the supporting library and environment.

Features of the Lattice C compiler for the 370 include:

- **Generation of reentrant object code.** Reentrancy allows many users to share the same code. Reentrancy is not an easy feature to achieve on the 370, especially if you use non-constant external variables, but we did it.
- **Optimization of the generated code.** We know the 370 instruction set and the various 370 operating environments. We have over 100 staff years of assembler language systems experience on our development team.
- **Generated code executable in both 24-bit and 31-bit addressing modes.** You can run compiled programs above the 16 megabyte line in MVS/XA.
- **Generated code identical for OS and CMS operating systems.** You can move modules between MVS and CMS without even recompiling.
- **Complete libraries.** We have implemented all the library routines described by Kernighan and Ritchie (the informal C standard), and all the library routines supported by Lattice (except operating system dependent routines), plus extensions for dealing with 370 operating environments directly. Especially significant is our byte-addressable Unix®-style I/O access method.
- **Built-in functions.** Many of the traditional string handling functions are available as built-in functions, generating in-line machine code rather than function calls. Your call to move a string can result in just one MVC instruction rather than a function call and a loop.

In addition to mainframe software development, you can also use our new cross-compiler to develop PC software on your IBM mainframe. With our cross-compiler, you can compile Lattice C programs on your mainframe and generate object code ready to download to your PC.

With the cross-compiler, we also offer PLINK86™ and PLIB86™ by Phoenix Software Associates Ltd. The Phoenix link-editor and library management facility can bind several compiled programs on the mainframe and download immediately executable modules to your PC.

Tomorrow...

We believe that the C language offers the SAS System the path to true portability and maintainability. And we believe that other companies will make similar strategic decisions about C. Already, C is taught in most college computer science curriculums, and is replacing older languages in many. And almost every computer introduced to the market now has a C compiler.

C, the language of choice...

C supports structured programming with superior control features for conditionals, iteration, and case selection. C is good for data structures, with its elegant implementation of structures and pointers. C is conducive to portable coding. It is simple to adjust for the size differences of data elements on different machines.

Continuous support...

At SAS Institute, we support all our products. You license them annually; we support them continuously. You get updates at no additional charge. We have a continuing commitment to make our compiler better and better. We have the ultimate incentive—all our software products depend on it.

For more information...

Complete and mail the coupon today. Because we've got the development tool for your tomorrow.

SAS®

SAS Institute Inc.
SAS Circle, Box 8000
Cary, NC 27511-8000
Telephone (919) 467-8000 x 7000

I want to learn more about:

- ☐ the C compiler for MVS software developers
- ☐ the C compiler for CMS software developers
- ☐ the cross-compiler with PLINK86 and PLIB86

today...so I'll be ready for tomorrow.

Please complete or attach your business card.

Name ______________________

Title ______________________

Company ______________________

Address ______________________

City ______________ State ________ ZIP ________

Telephone ______________________

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA. 27511-8000. Telephone (919) 467-8000, x 7000

DDJ 3/87

(continued from page 110)

(*int 24h*) handler. The more recent newsletters range from 6–11 pages in length and have a heavy emphasis on working source code. Subscriptions cost $18 per year. Contact Mr. Claff at 7 Roberts Rd., Wellesley, MA 02181; (617) 235-9505.

Call for Papers

The Waite Group, a San Francisco-based computer book developer and publisher, is looking for contributing authors for a new book on MS-DOS entitled *The MS-DOS Papers*.

The news release from the Waite Group says: "*The MS-DOS Papers* will be a collection of learning tutorials written by a broad range of MS-DOS experts, gurus, wizards, and spokespersons. *The MS-DOS Papers* will provide insightful information on the MS-DOS operating system, revealing the more hidden and obscure truths about MS-DOS in an interesting, easy to read Waite Group format. Its contributed nature allows us to include subjects that might not support a separate book as well as subjects that are on the cutting edge of MS-DOS technology. The audience level is intermediate to advanced businesspeople, programmers, and anyone who wants the most up-to-date information about this popular operating system. Examples are given in both MS-C and MASM.

"The book will consist of three types of contributions:

- Tutorials on topics that have never been adequately discussed in the literature. These include inside BIOS, tips and undocumented secrets, stay resident programming, advanced MASM programming, and debugging as well as new concepts arising in MS-DOS, such as protected mode MS-DOS and CD ROMs.
- Issue papers by experts in a particular area of MS-DOS. These will discuss past controversies, the future of MS-DOS, and so on.
- Case-history papers, which will tell the bottom line about real MS-DOS machines, projects, and software tools."

For more information, contact Mitchell Waite at one of the following electronic mail addresses:

BIX: mwaite
The WELL: mitch
Usenet: lll-lcc, hplabs}!well!mitch

A Nifty Tool

Cruise Control is a Terminate and Stay Resident (TSR) utility for IBM PCs and compatibles that eliminates *cursor runon*, the term the utility's author uses for the behavior of programs that cannot process keystrokes as fast as the keyboard's auto-repeat rate. When you are using such a program and hold down a key, the keystrokes pile up in the type-ahead buffer until it is full and then you hear a beep that tells you to release the key. The piled-up keystrokes are then processed until the buffer is empty, so you frequently tab or scroll much farther than you intended to. Lotus 1-2-3, WordStar, and Microsoft Word are three commonly used programs that have this problem on older 8088 or 8086-based PCs.

The primary effect of Cruise Control is that it monitors the type-ahead buffer and dynamically adjusts the

Name
PASTE—horizontally concatenate two files
Synopsis
paste [−paste] [−b <string>] [−<n>] [file1] [file2]
Description
PASTE will append to the lines of *<file1>* the corresponding lines of *<file2>*, with an optional string between them. PASTE writes to standard output.

The following flags are recognized by PASTE:

−p	*<file1>* does not exist (*<string>* is prepended to each line).
−a	*<file2>* does not exist (*<string>* is appended to each line).
−s	Do not print *<string>* with lines from only one file.
−t	Resolve the ambiguous command *paste <file>*. The *−t* flag forces *<file>* to trail standard input—that is, *paste <file>* is equivalent to *paste <file> <stdin>*, and *paste −t <file>* is equivalent to *paste <stdin> <file>*.
−e	Do not print *<string>* if both input lines are empty (contain no characters but '*\n*').
−b	Indicates that a string of characters follows. The string is inserted between each line of *<file1>* and *<file2>*. The string can contain all the standard escape codes with the exception of '*\0*'. The escape sequence '*\s*' is also known to represent a blank. Blanks may also be embedded in a string by enclosing the string in quotes.
−<n>	Print *n* lines of *<file1>* before appending lines of *<file2>*. If *n* is negative (for example, *paste −−3*), then *n* lines of *<file2>* will be printed first.

Bugs
On some systems, you'll have to use an escape sequence to represent capital letters in *string*. Also, a quoted string with multiple blanks can have them reduced to single blanks on systems that do not recognize quote marks as special—use the escape sequences '*\s*' or '\ '.

As of this writing, the standard escape sequences are:

\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	tab
\0	null character (not allowed in string argument)
\\	literal backslash
\"	literal quote mark
\'	literal apostrophe
\ddd	bit pattern, consisting of 1 − 3 octal digits

Escape sequences special to PASTE:

\s	space

A backslash followed by any other character merely represents that character.

Author
John M. Gamble, January 1984

Table 1: *Instructions for using the PASTE utility*

keyboard auto-repeat rate to match the program's capability to process the keystrokes. This means that you never tab, page, or scroll past your desired destination. For those programs that can handle it, the apparent speed of many keys (such as the arrow or page keys) is drastically increased.

It sounds like a simple concept, but the difference in the behavior of your computer and favorite editor with Cruise Control installed is dramatic. I have used it with both Microsoft Word and MicroPro's WordStar with excellent results. Cruise Control also offers a few nifty fringe benefits, such as an automatic screen dimmer after a configurable time delay, online help, and a date and time stamper with configurable formats. The vendor claims that the utility is compatible with most other RAM-resident programs; it worked fine for me with both SideKick and ProCED.

You can obtain Cruise Control from Revolution Software Inc., 715 Rte. 10 E, Randolph, NJ, 07869; (201) 366-4445.

Programming Pearl of the Month

Richard Rodman, of Falls Church, Virginia, writes: "Here's a helpful hint for programmers attempting to write adapting I/O routines.

"The IBM PC data bus is not pulled up. If you try to read a data port to see if the board is or is not installed, and the board is not installed, you may get a false indication because the floating bus still contains the last data byte that was fetched by the CPU.

"To correct this problem, you need to ensure that the bus contains a pattern with as many bits set to 1 as possible. One method of doing this is shown below:

```
clc
clc
clc
clc
clc
clc
in al,dx
clc
clc
clc
clc
clc
clc
```

"The 8088's instruction prefetch queue is 6 bytes long. The six *clc* instructions (opcode *0f8h*) on each side of the *in* instruction allow the bus to float at a value of *0f8h* hex for unimplemented hardware.

"The real solution, of course, would have been a terminated bus. Unfortunately, the IBM PC was a quick and dirty design."

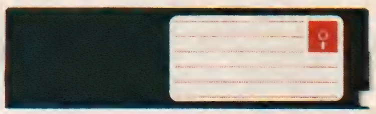
The PASTE Utility

John Gamble of West Lafayette, Indiana, has sent in a useful program called PASTE that appends the lines of one file to the end of the lines of another file and writes the resulting lines to the standard output device. PASTE can be used to horizontally concatenate tables or columns of information that have been edited separately. The program optionally prepends, appends, or inserts a string into the newly generated lines. Table 1, page 112, contains instructions for using the program, and the program's source code accompanies this column as Listing One, page 78. I have tested the program before publication with Microsoft C, Version 4.0, and MS-DOS, Version 3.1.

John writes: "I have found this pro-

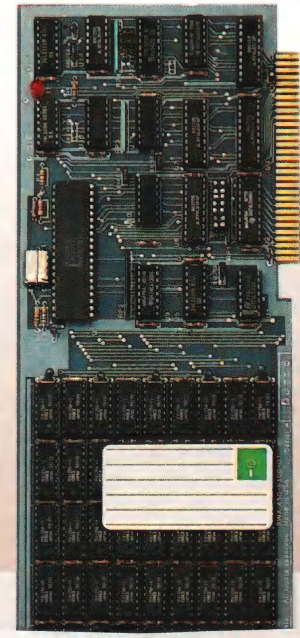
Think fast! Pick the better fit...

Our 5th Year Bonus!
Mention this ad when ordering
and get your choice of a V 20-8
(replaces 8088) or $20 off on
your Battery Backup purchase!



FLOPPY DISK.

- Fills time between coffee breaks
- Makes a hard disk seem *fast*
- Your computer appears busy (even if you aren't!)
- Wears out moving parts



SEMIDISK Disk Emulator.

- Gets that job done NOW
- Makes a hard disk seem *slow*
- Maximizes your productivity with anything from databases to compilers
- Totally silent operation

...for YOUR demanding tasks.

SURPRISE! *Neither* is memory mapped, so they don't affect your precious Main Memory. *Both* retain data indefinitely - even with the computer turned off.

THE SEMIDISK SOLUTION. You could invest in a series of "upgrades" that turn out to be expensive band-aids without solving your real problem. Even those "Accelerator" and "Turbo" boards do little to speed up disk-bound computers. If your applications spend too much time reading and writing to disk (and whose don't?), you won't want to settle for anything less than a SemiDisk disk emulator. The SemiDisk comes in 512K and 2Mb capacity. More boards may be added to make up to an 8 Megabyte SemiDrive!

SPEED THAT'S COMPATIBLE. PC, XT or AT, if you need speed, the SemiDisk has it. How fast? Recent benchmarks show the SemiDisk is from 2 to 5 times faster than hard disks, and from 25% faster (writing) to several times faster (random reads) than VDISK and other RAMdisk software that gobble up your main memory.

MEMORY THAT'S STORAGE. Using our small external power supply, with battery backup, your data remains intact through your longest vacation or even a seven-hour power failure!

CELEBRATE WITH US! Now, SemiDisk celebrates its fifth birthday with a special offer for IBM-PC owners. Buy a SemiDisk now and we'll include an 8 MHz V-20 microprocessor (replaces the 8088) to make your new SemiDisk run even faster. Don't need the V-20? We'll take $20 off the price of your Battery Backup Unit!

	512K	2Mbyte
IBM, PC, XT, AT	**$495**	**$ 795**
Epson QX-10	**$495**	**$ 995**
S-100 SemiDisk II	**$795**	**$1295**
S-100 SemiDisk I	**$299**	-------------
TRS-80 II, 12, 16	**$495**	**$ 995**
Battery Backup	**$130**	**$ 130**

**Someday you'll get a SemiDisk.
Until then, you'll just have to...wait.**

SemiDisk

MasterCard VISA

SemiDisk Systems, Inc., 11080 S.W. Allen Blvd., Beaverton, Oregon 97005 (503) 626-3104

(continued from page 113)

gram useful for creating command files on the fly for systems such as VMS or MS-DOS by taking a directory listing as an input file and attaching strings to the beginning or end of the lines. The Unix shell can do this by itself, but there are still some tricks that can be played with PASTE. For example, the command

```
paste -a -b '\n' afile.txt
```

will double-space the lines in afile.txt.

"I think that, if the program needs any improvement, it is in its method of input/output—it is done character by character. I really ought to have made it more efficient, but I fell foul of the 'good enough' syndrome and lost interest.

"[After writing this program] I learned that there is a similar Unix System V utility, also called PASTE. It appends the lines of one file to another, too, but it automatically inserts a tab between the lines and will accept more than two files on the command line—I think it is meant more for nroff text processing."

Semester Final

Larry Heberlein, of Maryville, Maryland, submits this little tidbit:

"Final Exam: Algorithm Design 101
Extra-Credit Question
You attempt a search and replace operation using a commercial word processor—the latest version of Word from Microsoft, the world's largest microcomputer software house. . . . You load a 20K file into a PC with 640K RAM. With the program and file loaded, 400K of memory are free. You attempt to replace every carriage return in the file with a space. Less than a quarter of the way through the file, the operation aborts with the error message 'insufficient memory.' You observe that this happens reliably, in any file, on any replacement, with a sufficiently large number of occurrences.

"An A for the course goes to any student who turns in a replacement algorithm so bad that it can't succeed in memory 20 times the size of the data."

Assembly vs. High-Level Languages

Charles Lyall of Kingman, Alberta, writes: "I couldn't let your invitation in the July 1986 issue of *DDJ* to discuss the assembly-language vs. high-level-language issue go unanswered.

"I am an EDP consultant who hacked his first piece of code in 1963 on an IBM 1620. Even in those days we were arguing the relative merits of assembly vs. higher-level languages. Now we have several fourth-generation languages that help to spice up the debate even more.

It is not always true that assembly-language programs run faster than high-level programs.

"I must take issue with your statement that 'It doesn't take me more than an hour or two to write a program the size of TEE from scratch in assembly language. . . .' The statement is false! Oh, I am quite sure that you could write it in Microsoft MASM for the IBM PC. Could you write it in two hours in assembly language for the VAX? No, then how about a Data General NOVA? A UNIVAC 1100 perchance? I think not. I can keep two assembly languages in my head at one time, but that is it. I doubt if you are truly fluent in more than two assembly languages either.

"What you really meant is that you can write TEE in assembly language for one particular machine in two hours. Mr. Gary Woodman can write the equivalent program in a few minutes for every machine that has a C compiler. I suggest that the C compiler is infinitely more productive.

"To quote you again: 'For me, the benefits of the superior performance and compactness of an assembly-language program almost always outweigh all other considerations for utility programs I am going to run more than once.' I submit that this is illogical. Let us put some numbers on it, Ray. Suppose Gary Woodman's program runs in 10 seconds and your program runs in 1 second. But you took two hours to write your program, and Gary probably took 15 minutes. That's a difference of 105 minutes, or 6,300 seconds, of coding time. At a 9-second advantage per run, you are going to have to run the little turkey 700 times in order to break even on total elapsed time.

"It is not necessarily always true that assembly-language programs run faster and take less space than equivalent high-level-language programs. A case in point is a little routine available to strip the high-order bit from WordStar files. It is written in assembly language and handles its input and output one character at a time, and it uses DOS to redirect both input and output. It is slowwww! I wrote a C program that reads in 16K of input, strips the bit off using register variables for my pointers to make things trot along, and then writes the buffer. Now that moves. True, my routine is much bigger, but in this case I will cheerfully trade size for speed. It took only a few minutes to write, too.

"Your emphasis on performance and compactness is not without merit, and I can argue your side of the debate, too. The microcomputers we have had to deal with in the last ten years have been characterized by very limited memory, poor CPU performance, and expensive slow backing storage. Under these circumstances you can raise a heck of a good defense for your position. But this situation is coming to an end. A half megabyte of memory is now common. Processors such as the 80286 can almost get out of their own way, and the latest generation of 68000 chips are quite peppy. The 80386 machines will probably accept 16 megabytes of directly addressable memory (a 24-bit memory bus) and be two or three times as fast as the AT machines. [*The 80386 can actually address 4 gigabytes of physical memory and some 70 terabytes of virtual memory.—Ray*]

"Look what happens now to the numbers I gave you a few paragraphs back. If the machine is three times as fast, then your utility will run in 0.3 seconds and Gary's will run in 3.3 seconds. It will now take 2,100 executions of the utility before you

Yes! I want to subscribe to
Micro/Systems Journal
and save over 15% off the cover price.

___ **1 Year (6 issues) $20** ___ **2 Years (12 issues) $35**

___Please charge my: ___ Visa ___ MC ___ American Express
___ Payment enclosed ___ Bill me later

Card # ________________________ Exp. date ____________
Signature ________________________
Name ________________________
Address ________________________
City ________________ State ______ Zip________

Canada & Mexico add $3 for surface mail; $7 for airmail. All other countries add $12 for airmail. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6–8 weeks for delivery of first issue.
A Publication of M & T Publishing, Inc.

300

Yes! I want to subscribe to
Micro/Systems Journal
and save over 15% off the cover price.

___ **1 Year (6 issues) $20** ___ **2 Years (12 issues) $35**

___Please charge my: ___ Visa ___ MC ___ American Express
___ Payment enclosed ___ Bill me later

Card # ________________________ Exp. date ____________
Signature ________________________
Name ________________________
Address ________________________
City ________________ State ______ Zip________

Canada & Mexico add $3 for surface mail; $7 for airmail. All other countries add $12 for airmail. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6–8 weeks for delivery of first issue.
A Publication of M & T Publishing, Inc.

30

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

For the Advanced Computer User

Micro/Systems Journal™

501 GALVESTON DR.
REDWOOD CITY, CA 94063

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

For the Advanced Computer User

Micro/Systems Journal™

501 GALVESTON DR.
REDWOOD CITY, CA 94063

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

For the Advanced Computer User

Micro/Systems Journal™

501 GALVESTON DR.
REDWOOD CITY, CA 94063

Subscribe to Micro/Systems Journal

(continued from page 116)

break even! The faster a machine is, the less benefit assembly-language code is on that machine.

"A similar argument can be made about storage. The cheaper mass storage and memory become, the less advantageous compact code becomes. Accountants call it a rise in the opportunity cost. In other words, by writing a utility in assembly language, you forego the other utilities that you could have written if you had used a more productive language. Against this you balance benefits of program size and run time, which become less and less significant as computer speeds rise and primary and secondary storage costs decline.

"As a high priest, to use Jerry Pournelle's epithet, I use assembly language to answer two classes of problems: when I can't describe the procedure in a high-level language and when a critical small portion of the program runs too slowly.

"With the advent of true fourth-generation languages, your position is going to be even harder to defend. I am currently designing a system using Powerhouse, a fourth-generation language for superminis. In less than three hours, I can generate a procedure to paint a screen with a form, accept any number of fields from that screen, and update a record or create a new record in a file that has three indexes. The generated procedure is about 8K long. On a VAX machine, the run time is sufficiently close to zero that it isn't material. I have no doubt that a good macro assembler programmer could write a similar routine in about a week. The resulting routine costs the client $150 if I write it in three hours. The same routine written in assembly language is not worth a penny more. The Powerhouse code, which is nonprocedural, will be trivial to maintain.

"Interestingly enough, my company has encountered a problem with a communications handler, and the communications expert, a red-hot macro programmer, intends to solve that one in FORTRAN. Certainly he could solve it in assembly language, but he can do it cheaper in FORTRAN. On a 16-megabyte VAX 780, the run time and space disadvantages are immaterial.

"In the long run, we spend three times as much time and effort on code maintenance than we do on the initial design and coding. To me, the benefits of clarity and simplicity in code and ease of maintenance outweigh all other considerations about 90 percent of the time. The other 10 percent of the time, we are dealing with the nasty bits that shouldn't be discussed in a family magazine.

"In conclusion, my position is that languages are tools to be used to solve problems and they are not ends in themselves. They have merits only to the extent that they help us meet our objectives. Two of these objectives are speed and compactness. Other objectives are clarity, simplicity, self-documentation, maintainability, programmer productivity in lines per day, and so on.

"A person who uses only one programming language puts me in mind of the man whose only tool is a hammer. All his problems look like nails."

Thanks, Charles, for a beautifully written, educational, and witty letter. I wouldn't want you to go away believing that my only tool is a hammer; I use Forth, C, and even BASIC (I hope that at least one of those meets your criteria for a high-level language). But I feel most at home with assembly language, and contrary to your assertion that it is not possible to be fluent in more than two assembly languages, I consider myself quite fluent in 8080, Z80, 80x86, PDP-11, and Raytheon 703 assembly language. I can also get by well enough in 6502, 8051, 8096, 68000, and 1802 assembler language. But I admit to total ignorance of the UNIVAC, Data General, and VAX that you mentioned!

Availability

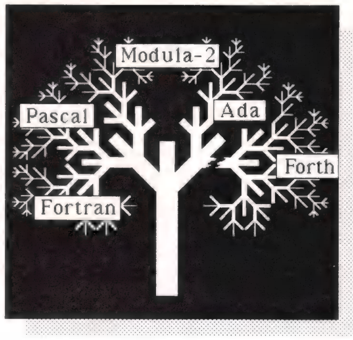
All the source code for articles in this issue (except for C Chest) is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listing begins on page 78.)

Vote for your favorite feature/article.
Circle Reader Service **No. 5**.

BASIC: Quo Vadis?



In this issue, I discuss the differences between the IBM PC BASICA and the new True BASIC and QuickBASIC. I also include a short utility program to show interdialect translation and discuss several differences in these BASIC dialects.

In most cases, BASIC is the language that microcomputer users learn first, and in the IBM PC world, the implementation they encounter is Microsoft's BASICA. BASIC has been judged inadequate for large software projects, difficult to maintain, and lacking many new programming concepts. With the advent of structured languages, such as Turbo Pascal, programmers have been given the taste of better techniques, and the myth that BASIC will always be *the* language is no longer true.

New BASIC Dialects

The wheel of progress has not spared BASIC from change, however. Two years ago, the original authors of BASIC (Kemeny and Kurtz) launched True BASIC, a more structured implementation that is close to the new proposed ANSI BASIC. Almost simultaneously, Microsoft launched a new BASIC compiler version, QuickBASIC, and in mid-1986, it introduced Version 2.0, which includes a versatile environment. QuickBASIC is not just another compiler for BASICA—it brings with it a new syntax, similar in many instances to that of True BASIC.

by Namir Clement Shammas

QuickBASIC does not require line numbers; instead, you place alphanumeric labels in your programs to direct branching. True BASIC, however, requires the entire program either to have line numbers or to have none at all. If you use *GOTO* or *GOSUB* in True BASIC, you need the line numbers; otherwise, they are not mandatory. To my disappointment, True BASIC does not support labels.

Both True BASIC and QuickBASIC support more structured program code. Multiline functions and subroutines (with argument lists) enable you to create more modular code that is easier to maintain and enhance. The new dialects also implement external libraries with the added notion that not all variables are global, which resembles many features in FORTRAN.

What about translation between BASICA and True BASIC or QuickBASIC? As you may expect, because Microsoft wrote both BASICA and QuickBASIC, the two dialects have many built-in functions and statements in common. As a rule of thumb, the aspects of BASICA not available to the QuickBASIC compiler, such as *CHAIN MERGE*, are related to the interpreter features. In general, BASICA is upward compatible with QuickBASIC.

Translating programs from BASICA to True BASIC requires more work. True BASIC Inc. sells a BASIC converter to handle many systematic conversion steps. Like the Logitech Translator I discussed in my last column, the BASIC converter does not translate 100 percent of the BASICA code. Later I will present a sample BASICA program and its translated versions in True BASIC and QuickBASIC. To pave the way, I will first discuss several differences between the three dialects.

Similarities and Differences

Concerning data types, BASICA and QuickBASIC support an identical set of strings, integers, and single- and double-precision reals. True BASIC supports even simpler types: strings and numbers. The distinction between integers and reals is context-sensitive—if a number has no fractional part, it is stored as an integer; otherwise, it is stored as a real. String manipulation follows a different syntax in True BASIC. For example, when you extract and assign substrings in BASICA, you use something such as *MID$(L$,FIRST%,LONG%)*. In True BASIC this is written as *L$[FIRST:(LONG+FIRST-1]*. The square brackets and the colon inside them specify the first and last characters (as opposed to the number of characters in BASICA). True BASIC uses the ampersand to concatenate strings and names several string functions differently.

True BASIC supports matrix operators, functions, and I/O procedures. While hand-translating BASICA programs that perform matrix operations, you can substitute blocks of code lines with single *MAT* statements.

Loop constructs in BASICA and QuickBASIC are identical. I look forward to seeing more powerful *REPEAT . . . UNTIL*-like loops in the next version of QuickBASIC. True BASIC offers a variety of loops that include the *FOR . . . NEXT*, *DO*, *DO WHILE*, and the double-test *DO WHILE . . . LOOP WHILE* loops. If you perform manual translation, many BASICA logical loops with *GOTO*s can be rewritten using any of the True BASIC loops, which enhances readability.

Decision-making constructs in QuickBASIC and True BASIC are superior and far more readable than in BASICA. QuickBASIC and True BASIC support multiline *IF . . . THEN . . . ELSE . . . END IF* constructs and even allow for *ELSEIF* clauses. The *ON GOTO* and *ON GOSUB* use labels with QuickBASIC. QuickBASIC does not support the *CASE* statement, whereas True BASIC does. Translating BASICA *IF . . . THEN . . . ELSE* statements enables you to use the clearer multiline

version in the other dialects. Gone are the frustrating branchings in the *THEN* or *ELSE* clause that breathe chaos in your program. The *ON GOTO/GOSUB* are easily translated to the superior *SELECT CASE* statement in True BASIC.

How many times have you felt the limitations of BASICA in defining functions? How many times did you have to use a subroutine to simulate multiline functions? QuickBASIC and True BASIC make these painful memories a thing of the past. Now function definitions can extend over numerous lines and freely use loops and decision-making constructs. Whereas True BASIC supports recursive functions, QuickBASIC in its current version does not.

Regarding subroutines, both QuickBASIC and True BASIC support the *GOSUB <label or line number>* and *CALL <subname>* forms. The called subroutines take optional argument lists. Both QuickBASIC and True BASIC provide functions that return the lower and upper bounds of arrays. These functions are vital for writing general-purpose routines that manipulate arrays and matrices.

Both QuickBASIC and True BASIC support external libraries of routines. At the time of writing this column, True BASIC Inc. announced True BASIC, Version 2. One of its highlights is the introduction of modules! I will discuss True BASIC modules in my next column, once I obtain more information on the exact syntax and features. I will also discuss any aspects of similarity between library modules in True BASIC 2.0 and Modula-2. Although BASICA does not support explicit libraries, you may want to consider creating external libraries that contain your favorite and frequently used routines.

BASICA programs that use low-level features (*DEF SEG, VARPTR*) translate easily into QuickBASIC. True BASIC does not support such machine-specific statements, however, because they make programs less portable to other machines. For the same reason, the valuable *SHELL( )* statement found in both BASICA and QuickBASIC has no similar implementation in True BASIC. The Developer's Toolkit, offered by True BASIC Inc., does provide several low-level access routines for the IBM PC implementation.

High-resolution graphics is another area in which BASICA programs need more effort to be converted into True BASIC. I think that True BASIC's built-in graphics features are superior to those of BASICA. For example, True BASIC supports the *PICTURE* type of routines, special kinds of subroutines that make animation of objects easy.

File I/O is very similar in BASICA and QuickBASIC. True BASIC uses a slightly different syntax and organization, which means additional editing of converted BASICA programs. The *LPRINT* statement in BASICA is not supported by True BASIC. Instead, you must open a buffer for the printer (for example, *OPEN #<Buf_Num> : PRINTER*) and then send all the printer output using "*PRINT #<Buf_Num>*" statements, similar to file output. In translating BASICA programs, you must insert the *OPEN* statement and replace every *LPRINT* with "*PRINT #<Buf_Num> :* ".

Error handling in BASICA and QuickBASIC is also similar, both using the *ON ERROR GOTO* and *RESUME* statements. QuickBASIC uses labels to direct the program flow to error handling sections. True BASIC uses a different and more structured mechanism—a *WHEN ERROR IN . . . USE . . . END WHEN* construct. The code section suspected of generating errors is located in the *WHEN* clause and the exception handling code in the *USE* clause. By enclosing the suspected code portion in the *WHEN* clause, the extent of error trapping is most noticeable.

Interdialect Translation

Listing One, page 88, presents a BASICA utility program. The user types in the number and names of data files containing text. This is followed by several search strings, with the options of simply locating or replacing strings with others. The entire set of strings is used in text manipulation with each file. The program prints the text lines found or altered and writes back the text files to update them. The replace mechanism is fully automatic and has no query option.

Using the BASIC converter from True BASIC Inc., I converted the BASICA program. Listing Two, page 89, shows the True BASIC version after manual editing that was needed to make the program function. The converter inserts several lines at the beginning of the original BASICA listing. These include the use of the deflib.tru library, which contains True BASIC functions that clone certain BASICA functions, listed in lines 25 and 26. Three author functions are defined within the converted program. The *Eof( )* function is used by the utility. The *OPTION BASE 0* is also used and does not conflict with my program. Notice the following changes made either by the converter or by hand coding:

1. The original *DEFINT* declaration is rendered passive by converting it into a comment.
2. Each dot character used in the name of a BASICA variable is replaced with two underscore characters.
3. BASICA program lines containing multiple statements are broken down into single statements per line in True BASIC.
4. I inserted line 1945 to open a buffer for printer output. All BASICA *LPRINT* statements were flagged by the converter. I changed each *LPRINT* into *print #9* :.
5. The BASICA *SPC( )* function is replaced by *REPEAT$(" ",<number>)* to produce the same effect. Using *TAB( )* is another alternative.
6. The converter moved the BASICA *END* statement from line 3000 to the very end and replaced it with a *stop*. In True BASIC, there must be one and only one *end* statement at the end of the program. If I manually replace the *stop* with *end*, all the subsequent subroutines become external (the current *end* location makes them internal). The difference between internal and external subroutines is in the scope of variables. Internal routines access all the variables of the main program, but external routines do not. Library files containing nothing but external routines must begin with the keyword *EXTERNAL*.
7. I edited the *OPEN* statements for file I/O to add the *create old* clause, which indicates that the file must already exist.
8. I added the *erase #1* in line 9015. This erases the contents of the file before I write back to it. Unlike BASICA, True BASIC does not allow you to overwrite existing text, so you must erase a file before updating its

STRUCTURED PROGRAMMING
(continued from page 121)

contents.
9. Each assignment statement in True BASIC begins with the keyword *let*. It is mandatory in Version 1, but the new Version 2 enables you to issue a directive and make the *let* optional.

Listing Three, page 91, shows a True BASIC version that differs from that in Listing Two in the following ways:

1. No line numbers are used.
2. Some of the tests in the *IF . . . THEN* constructs have been reversed to make use of the multiline *THEN* and *ELSE* clauses and to bypass the need for line numbers.
3. Subroutine *CALL*s are used instead of *GOSUB*. I have deliberately used argument lists to give a sense of structured code. I could have made the subroutines parameterless and their code access global variables.
4. The BASICA *NOT EOF( )* test used in detecting the end of file is replaced with the True BASIC *MORE #1* function, which performs the same task.

Listing Four, page 92, shows the first QuickBASIC version of the BASICA utility. I wrote it to demonstrate the following QuickBASIC features:

1. No line numbers.
2. The *GOSUB* statements are followed by alphanumeric labels. The corresponding labels are located at the start of each subroutine. This QuickBASIC looks slightly more modular than its parent BASICA versions.

Listing Five, page 94, shows the second, more structured QuickBASIC version. The *GOSUB* statements are replaced by subroutine *CALL*s. The argument lists of the subroutines are identical to those of True BASIC in Listing Three. Listings Three and Five show strong similarities between QuickBASIC and True BASIC with respect to program segmentation. This gives you a feeling that both QuickBASIC and True BASIC really promote more structured code. Compared to Pascal, these BASIC dialects retain simple data types with the declaration of variables limited to arrays. Compared to FORTRAN, they represent a true challenge because they offer many of FORTRAN IV and FORTRAN-77's features.

I have focused on the one-way translation of programs written in BASICA to QuickBASIC and True BASIC. In a future column, I will look at the two-way translation between QuickBASIC and True BASIC programs. I also plan to look at Better BASIC, another "new wave" BASIC dialect, which I have not discussed this time because of space limitations.

Availability

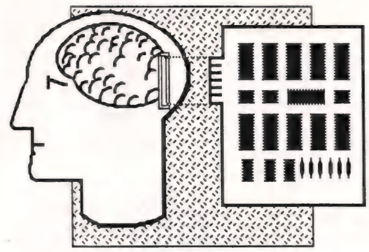
All the source code for articles in this issue (except C Chest) is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kapro).

DDJ

(Listings begin on page 88.)

Vote for your favorite feature/article.
Circle Reader Service **No. 6.**

Object-Oriented Programming



The theme for my next few columns will be object-oriented programming in AI. This is a rather vast but very hot topic, and I'll approach it from a few different vantage points. As mentioned in the previous column, object-oriented languages represent a programming paradigm, which is a much more significant development than just another new programming language or programming technique. Most programming languages to date have been for programming in a single paradigm, that of procedural programming. Because all programmers already know this paradigm, I will concentrate on the newer ones.

As I see it, object-oriented programming is not a direction that is entirely new and without precedent in computer science; rather, it takes various developments in programming languages to their next logical step, for reasons of clarity, modularity, and programming efficiency. In one sense, you can think of object-oriented programming as the programming paradigm that takes structured programming to its natural logical conclusion. In structured programming, variables can be local to a particular procedure and these procedures typically pass arguments such as strings and numbers between them. With object-oriented systems all this is taken much further. Variables are no longer local just to procedures. The main building blocks are now objects—protected areas of system memory—which can have both local variables and local procedures. Moreover, the building blocks do not communicate with one another just by passing arguments. The procedures themselves, usually called methods, which are local to objects, are actually the messages that are sent and received by objects. In this respect, objects resemble smaller computers within the host computer, each with their own data and code areas.

by Ernest R. Tello

Most object-oriented systems have at least two different types of objects: classes and instances. Classes may have a logical relation between one another such that one might be the subclass or superclass of another. Generally speaking, the superclass is the more abstract class and the subclass is the more specific. So, for example, if you created the class *Furniture*, then you could create the class *Chair* as the subclass of *Furniture* and *Desk-Chair* as a subclass of *Chair*. In this example, *Furniture* would be the superclass of *Chair*, which is in turn the superclass of *Desk-Chair*.

Object-oriented systems have at least three obvious advantages. One very nice one is that, once you have written the code for a class, you can have as many instances of that class present in the system at the same time as memory will allow. A class is simply a template on which each instance is modeled and provided with its own area of memory that cannot be accessed by any other object except by using the object's own local methods. So, for example, this means that, in an object-oriented system, you can have as many graphics pens, windows, editors, interpreters, and so on as you like copresent without any fear that they will interfere with one another. The second advantage is that, through the mechanism of inheritance, subclasses automatically share all the variables and methods of their superclasses. This means that you can write greater and greater specializations of functions just by adding the part that is unique—the rest is inherited automatically. The third immediate advantage is that you can provide a uniform interface over the widest possible range of object types because you can use the same name for methods of different objects that have to be implemented differently, and this action can remain invisible to the user. So, for example, you might create different classes for a variety of different geometric polyhedra. Then, for each separate class, you would define the methods volume and surface area. The actual formulas and their implementations would vary, but the calling names would all be the same. Then you could say *Tetrahedron-1* volume or *Cube-3* volume, and in each case methods would be invoked that returned the value of the object's volume.

Some people say that the key advantage of object-oriented programming is the ability to reuse code for many different programs. But in itself this is not significantly different from library functions. The real difference is an improved ability to handle complexity in a transparent manner. An advantage of object-oriented programming that is not necessarily immediately obvious—but which experienced programmers who have worked with these systems will nearly always testify to—is that object-oriented languages give you more leverage in working on very large programs. This does not come for free, though, and it's not guaranteed. Factoring a large program into the right parts is a large part of the battle. It is also necessary to learn the right techniques for managing the code and making life easy for the members of a programming team. Object-oriented systems are usually diffuse,

(continued from page 126)

with parts of applications being dispersed among a large number of classes and subclasses. To program efficiently with such a system, it is essential to have the proper tools and an effective method for keeping the application well focused and well organized.

It is important to point out that object-oriented programming cannot be regarded as an easy thing to pick up rapidly, as it is a totally different paradigm from the one to which most programmers are accustomed. As programming approaches go, it is a knowledge-intensive one. In other words, the readily available modular code is only useful providing that programmers know what they have available to them and how it may be best used. Many programmers resist learning new languages, not to speak of new programming paradigms, so it is important to spell out again in clear-cut, pragmatic terms just what the real advantages are to programming with objects. As I see it, there are four main ones:

1. standard calling conventions for a broad range of operations that exhibit differences in behavior, as do variations on a theme
2. a means of managing very large programming projects by breaking up large problems into smaller, independently functioning, highly visible parts
3. a truly modular programming environment in which redundancies in coding are kept to an absolute minimum
4. the ability to spawn multiple instances of a given function or object from the same code without the codes for the instances interfering with one another

Object-Oriented LISP: An Overview

Although Smalltalk was the first true general-purpose object-oriented language, and implementations such as Digitalk's Smalltalk/V are specifically aimed at AI applications, the main uses of this programming paradigm so far in AI have been with object-oriented LISP. The reason for this is probably because people in the AI field are most familiar with LISP.

The most interesting things that have occurred with object-oriented LISP are some of the clear innovations it has made in object-oriented programming generally. Not only did LISP have little difficulty absorbing the object-oriented paradigm but also it introduced some important innovations to this programming approach as it did so. I would like to discuss three innovations in particular—mixins, method combination, and multimethods. The mixin feature is the LISP version of multiple inheritance, which consists of the ability to create a new class that inherits from more than just a single superclass. In effect, it means the ability to build an object hierarchy that can be a network or tangled hierarchy rather than just a simple tree. Although multiple inheritance is theoretically present in the latest release of Smalltalk-80, it was an afterthought in this language and has nothing close to approaching the readily usable and trouble-free operation of mixins in object-oriented LISP.

Method combination is a bit more difficult to explain than mixins but is no less important. The first appearance of user-defined method combination in LISP was with Symbolics' Flavors system. This system does not just copy the approach to method combination used in Smalltalk but introduces a new approach. The Flavors implementation lays stress on the order in which components are combined to produce a flavor. This is particularly true with methods, the procedures that are local to flavors. The heart of the Flavors system is the way the methods of various components are combined. The problem is this: if you define a flavor that inherits from several other flavors or classes, each of which have their own specialized versions of the same message, then how will the method for this new flavor be constructed? The Flavors system offers a variety of ways for combining methods and even provides for user-defined method combinations. It is designed so that, if you want to, you can define entirely new ways of combining methods.

The default for method combination in Flavors is to ignore all but the latest implementation of the method, meaning the one defined in the most specific of the flavors from which the new flavor will inherit. If you decide to define an entirely new method for the new flavor, then all the others will naturally be overridden. The general format for the more complex types of method combination in Flavors is for one flavor to be selected to provide the primary method and for any other flavors to provide what are called daemon methods. The primary method has control of handling the main function associated with the message, whereas the daemon methods are responsible for subsidiary tasks.

Flavors has two kinds of daemon methods—*before* and *after*. The terminology is derived from the order in which the method functions are called. The basic way that combined methods work is that they first call all the before methods, then the primary method, and finally all the after methods. Each of these component methods is passed the same arguments in turn as were passed to the combined method. Only the values returned by the primary method will be returned by the combined method, however. All values returned by the daemon methods are ignored. If there is more than one before method, then the before methods are called in the same order as that in which the flavors are combined, whereas after methods are called in reverse order.

What is the point of these method combinations? They can have a variety of different uses, but one of the most obvious is to provide an additional type of modularity that captures the whole spirit of the object-oriented approach. With method combination, if you can't find all you want for a flavor method in any of the flavors from which it will be inheriting, yet part of what you want is available, then method combination can often save you from having to rewrite the entire function from scratch. What you can do is select a method to inherit that can serve as your primary method. Then you just write the before and after methods that can be added to this primary method to produce the desired result. Naturally this won't be possible in all cases. It will work only in situations in which the function desired

can be combined from several separate functions. You will find that this applies to a surprisingly large number of cases, however.

Multimethods, a capability first made available in CommonLoops, could well be the most important contribution object-oriented LISP has made so far to object-oriented programming. Multimethods are functions that can be considered as messages to any number of types of objects. Prior to the development of multimethods, object-oriented LISP made a distinction between the object to which a message was sent and the arguments to the message procedure. So, in the expression:

```
(send Rectangle draw-at 10 40)
```

the class *Rectangle* is solely responsible for recognizing the message and is distinguished from the numbers *10* and *40*, which are arguments to the *draw-at* message. This is somewhat artificial, for in Smalltalk numbers are treated as instances of the class *Number* and arithmetical operations such as multiplication are considered as messages to the numbers to multiply themselves. Multimethods take this even further by considering the class to which a message is sent as just another one of the arguments. In principle any number of arguments can be passed to a multimethod, and each of the arguments is an instance of its own class. So a multimethod is really a message to an indefinite number of objects, with the method combination required to complete this message determined by the actual arguments used.

One "problem" with object-oriented LISP is that it is just too popular for its own good. It has become a favorite tool not only for programming AI applications but also for systems programming and developing user interfaces. The result of this diversity is a somewhat conflicting set of requirements for users of different types. Systems programmers and those developing user interfaces and advanced graphics applications are usually interested in high-performance, bug-free code. AI researchers are willing to trade some performance for greater flexibility and generality.

What are some of the primary con-

(continued from page 129)

siderations in using object-oriented LISP for AI purposes? As I have already suggested, certainly one of the most important is the dynamic behavior of class systems. This refers to the ability of an object system to change dynamically to reflect changing circumstances in the world. In many AI programs it is of considerable importance that the system be able to update itself automatically to a greater or lesser degree. It is helpful to consider what this implies.

A minimum condition for such an automatic system is to keep a running tally of all the system's current objects of various kinds in a form that is easily accessible. In LISP this generally means maintaining a list of such items and being able to update the list as necessary. More specifically, it is necessary to be able to access at any given time all the instances that are currently alive and know their classes. If the system does not already do this in some way, then it is essential that it at least support the minimum functions that would allow these features to be implemented.

Closely related to this requirement is the ability to write functions that can create new objects with names that are determined only at run time. Although this may sound trivial, in LISP it is easy to create new objects by programming them with names the programmer specifies in the code, but it is not nearly as straightforward to write functions that automatically create objects when needed with names that have to be specified at the time. Compared with this, "uncreating" objects is relatively trivial. If there is no function corresponding to *remob* in use, then it is still always possible to make a new object with the same name as the uncreated one and set it equal to nil.

Such functions are necessary for creating what are known as composite objects, for example. These are objects of a complex structure that contain other objects as parts. So, for example, a *desk* object could be described as a composite object composed of a *top*, *legs*, and *drawers*. In such a case, the components might well be instances of classes in their own right. To create a composite object automatically, therefore, would involve naming and creating all those instances that are parts of the composite.

In a sense, the parts of a composite object can form another hierarchy that can exist alongside the abstraction hierarchy of classes containing the main composite object. The composite object approach seems to have real limits as far as creating very large hierarchies is concerned, however. It is difficult to imagine creating very large systems such as spacecraft in any degree of detail. Are such systems necessary? If you want to be able eventually to create deep systems for diagnosing problems and predicting various consequences in emergency situations, or even for failure-mode analysis for design purposes, then such systems appear to be indispensable.

Many LISP programmers are anxious for an object-oriented extension to Common LISP and wish there were already such a standard for the dialect. At the present time several LISP vendors have developed their own proposed extensions, which they hope will be adopted as the standard object-oriented extension to Common LISP.

In the Spotlight

I asked some experts in object-oriented LISP both what they wanted to see happen and what they thought would happen in the quest to develop an agreed-upon standard for object-oriented programming in Common LISP. Currently, there is an organized effort to develop such a standard with formal ANSI recognition. Toward that end, a committee has been formed to draft a proposed standard that will then be made available to the programming community for its feedback so that a true "community standard" might emerge. The members of this object-oriented working group include Dan Bobrow and Gregor Kiczales of Xerox; David Moon, Dan Weinreb, and Sonja Keene of Symbolics; Richard Gabriel and Linda Demichael of Lucid; Jim Kempf of Hewlett-Packard; and Patrick Dussud of Texas Instruments.

Gabriel is the president of Lucid, a vendor of Common LISP for a variety of different machines, and author of the book *Performance and Evaluation of Lisp Systems* (MIT Press, 1985). I spoke to him regarding the developing standard for an object-oriented extension to Common LISP. Gabriel sees his own role as a kind of "buffer zone" to help mediate the potentially inflammatory relations between Xerox, advocate of CommonLoops, and Symbolics, advocate of New Flavors. Although Gabriel is not optimistic about the solution of many of the subtle problems in formulating an adequate standard rapidly, he feels that a reasonable standard is emerging that has many of the features of New Flavors "on the surface" while utilizing much of CommmonLoops "underneath."

I asked Gabriel about some of the difficult issues involved in formulating the new standard, such as the issue of dealing somehow with the fact that readable code is often not efficient and, conversely, efficient code is often not readable. Gabriel admits that this is a pervasive problem that will not go away easily. One difficulty is that often programmers who know the details of a given application can find various ways to exploit certain idiosyncrasies of the way it is coded to improve performance considerably. Tricks of this kind are obviously specific to the particular application and are often opaque to another programmer reading the code. The main reason for this, according to Gabriel, is that "most efficient code uses side effects to a considerable degree." Gabriel thinks that advanced knowledge-based systems for program optimization will ultimately be necessary to solve this problem. Another possibility is parallel LISP because, as Gabriel points out, "the programs with the least side effects run fastest on parallel machines." Gabriel is currently working on Qlisp, a parallel LISP language, with John McCarthy, the inventor of LISP.

Another problem specific to object-oriented systems is that currently instances in most systems are strictly subservient to classes in that an instance object is always an instance of one and only one predefined class, which is used as a kind of template for creating the instance. This has a built-in bias for the abstract over the concrete, which could put a limit on

the type of object-oriented AI programs that can be written. For example, it might be highly desirable in certain AI programs that an object be created that is not initially an instance of any particular class but that later on might be. One way around this would be to first make the object an instance of a neutral holder class, such as *Object* in Smalltalk or *t* in CommonLoops, and have a provision for changing the object's parent class at a later time.

Another person I asked for comments about the object-oriented extension to Common LISP was Gerry Barber, chief scientist at Gold Hill Computers, the main vendor for a subset of Common LISP on MS-DOS machines. The main issues Barber and his group at Gold Hill seem concerned about are that the standard make use of those features that are well understood and trouble-free. In this respect, he has some doubts about the metaobject protocol that forms the heart of CommonLoops. Barber is apparently not as confident as members of the standards working committee are that the metaclass approach that works so well in Smalltalk and CommonLoops is well-tested enough in the LISP environment to find a place in the standard. He agrees that it would be desirable to have a system that has an inherent flexibility and generality, but "it is important," he says, "to find the right generality and the right flexibility." Barber sums up the outlook at Gold Hill as follows: "Our strategy is to rely on things that have been shown not to have problems." Regardless of whether the new standard is ready in time, Gold Hill plans to release its own object-oriented extension to Common LISP—a system that will probably closely resemble the Flavors system from Symbolics—early in 1987.

The third person I spoke to on the same topic was Dan Bobrow of the Intelligent Systems Laboratory at Xerox PARC, considered by many to be the foremost authority on object-oriented programming in AI. Compared with Gabriel and Barber, Bobrow is most optimistic concerning the emerging standard for objects in Common LISP. He doesn't feel that what has been reached so far is simply a compromise between the Xerox

ARTIFICIAL INTELLIGENCE
(continued from page 133)

and Symbolics proposals. He feels that what is happening is a genuine synthesis of the best ideas that are around right now in object-oriented LISP. To him, the atmosphere is not one of tension between competing proposals but rather a true professional collaboration, much in the same spirit that produced the original Common LISP standard.

I also asked Bobrow what some of the limitations might be with the emerging standard. Here again, his answers were positive. For example, I raised some of the issues I had discussed with Gabriel, such as the need in advanced AI applications to be able to have the same object simultaneously present in several different hierarchies. Here he felt certain that the new standard did not rule out the ability to program such applications. I also raised the issue of efficiency and trouble-free operation vs. adaptability and flexibility. Once again he was optimistic and indicated that it would indeed be possible to include different compilation options for different uses of objects. This would mean, for example, that those programmers using objects for systems programming and user interfaces, who are primarily interested in producing fast, unmodifiable, trouble-free code, could use one compilation and those who use objects in AI programs that need to have greater flexibility and the ability to modify themselves dynamically could use a different compilation option. In this way, both types of users could be satisfied.

In my next column I'll continue the theme of object-oriented AI by focusing on some specific implementations of object-oriented languages.

Bibliography

Bobrow, D., et al. "Merging LISP and Object-Oriented Programming." *OOPSLA '86 Proceedings.*

Moon, D. "Object-Oriented Programming with Flavors." *OOPSLA '86 Proceedings.*

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 7.**

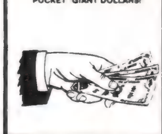
15 SUCCESS MANUALS that could solve your money problems once and for all!

FASTEST, EASIEST . . . PROVEN PROFITABLE BUSINESSES YOU CAN QUICKLY START AND OPERATE FROM HOME WITH LOW OR NO CAPITAL. . . PART TIME OR FULL TIME. . .

No experience required . . . Nothing complicated to study . . . Strictly legal and honest . . .
*Each beginners Success Manual is **Guaranteed** to teach you everything you need to know to succeed fast! The perfect answer for ambitious men and women . . .*

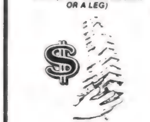
1. FIFTY QUICK, EASY AND MOST UNUSUAL WAYS TO POCKET "GIANT DOLLARS!"
Here's your chance to discover how so many folks miss out on numerous opportunities to pull in some big, fast cash. A most unique money-opportunity book which quickly shows you how just ordinary men and women from all walks of life are building spare time and full time fortunes; plus home businesses, money secrets, wealth-building methods, out-of-the-ordinary plans and odd blue prints to success, plus more. (only $6.95)

FIFTY QUICK, EASY AND MOST UNUSUAL WAYS TO POCKET "GIANT DOLLARS!"



2. HOW TO STACK UP HUGE MAIL ORDER PROFITS — HAND OVER FIST WITHOUT BREAKING YOUR BACK *(OR RISKING AN ARM OR A LEG)*
Shows you how to immediately set up — and get your operation off to a smooth flying start. Quickly teaches you short cut mail order fundamentals from A to Z. Crammed with insider "tricks of the trade" and revealing "money getting gimmicks." Imagine yourself receiving envelopes containing hundreds of dollars or more a day every day — that's the potential of mail order. (only $6.95)

HOW TO STACK UP HUGE MAIL ORDER PROFITS —HAND OVER FIST WITHOUT BREAKING YOUR BACK (OR RISKING AN ARM OR A LEG)



3. HOW TO SEW YOUR WAY TO PRETTY PROFITS FAST!
It's a fact that millions of women (and men, too) own their own sewing machines . . . and truly enjoy sewing. This peculiarly profitable book clearly demonstrates to them how to, virtually, turn their sewing machines into money making machines . . . and take fast and full advantage of today's most promising market conditions. Especially — considering the present sky-high prices. (only $6.95)

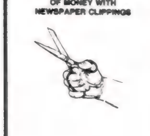
HOW TO SEW YOUR WAY TO PRETTY PROFITS FAST!

4. HOW TO TURN YOUR TELEPHONE INTO A MONEY MAKING MACHINE:
Right now your phone is only costing you money — but if you knew how to make it work for you, it could be making you money. Many people have heard about men and women making handsome incomes, via their telephone. But only a few people know exactly how it's done. Complete easy-to-follow instructions. (only $6.95)

HOW TO TURN YOUR TELEPHONE INTO A MONEY MAKING MACHINE

5. HOW TO EARN A FISTFUL OF MONEY WITH NEWPAPER CLIPPINGS:
Imagine, earning good money by clipping articles from newspapers? This unusual book instructs you in straight-to-the-point, how-to-information. Fast starting operation by mail on a tiny shoestring capital. Unusual way to earn $50, $100, $300 or more, weekly. Ideal for ambitious Homeworkers, spare-time or full. (only $6.95)

HOW TO EARN A FISTFUL OF MONEY WITH NEWSPAPER CLIPPINGS



6. HOW REAL ESTATE CAN MAKE YOU A FORTUNE . . . USING OTHER "FOLKS MONEY:"
Real estate has produced more millionaires than any other field. The plans inside this amazing fast, fortune-building book tells why and how, in easy-ABC fashion; learn how to let other folks money work for you; speculate in raw land and get back $5 for every $1 you put in; rake in huge profits on Uncle Sam's losses; set up a nice income for yourself and your family with little or no investment. (only $6.95)

HOW REAL ESTATE CAN MAKE YOU A FORTUNE... USING OTHER "FOLKS MONEY"

7. WORK-AT-HOME SUCCESS GUIDE (For Men and Women)
Time and time again — successful spare time and full time businesses are made with out-of-the-ordinary methods; off beat money making ideas, prosperous home enterprises. Shows how plain every day folks from all walks of life can stack up good money. Here's your opportunity to go after incredible wealth. (only $6.95)

THE AMAZING HOME WORKER'S WORK-AT-HOME SUCCESS GUIDE (For Men and Women)

8. AMAZING MONEY MAKING TREASURY OF 1 & 2 INGREDIENT FORMULAS THAT COULD PUT YOU ON EASY STREET
This startling opportunity book places the little "beginner" operator with tiny capital in a most profitable position to manufacture sellable products. All preparations require no more than two chemicals, many just one. All represent a popular best-seller kind of product with both genuine merit and wide sales appeal. No expensive equipment or facilities required. You can almost always pack everything from your kitchen. (only $6.95)

AMAZING MONEY MAKING TREASURY OF 1 & 2 INGREDIENT FORMULAS THAT COULD PUT YOU ON EASY STREET

9. WORLD'S EASIEST MOST PROFITABLE MAIL ORDER BUSINESS . . .
A relatively uncrowded business that any man or woman can enter regardless of age. Book shows you how to start small, with 'piggy bank' capital and grow prosperous year after year. Reveals the surest, most profitable and safest items to sell by mail. Crammed with all the precious, easy-to-understand details. (only $6.95)

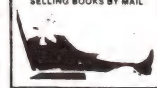
WORLD'S EASIEST MOST PROFITABLE MAIL ORDER BUSINESS

10. TWELVE SIMPLE LITTLE-KNOWN WAYS TO MAKE BIG MONEY FAST!
An amazing book that clearly reveals a dozen ways men and women could pocket some real fast cash profits — if they only knew the right wealth building moves to make. This book quickly teaches you all the necessary moves, shows you exactly how and what to do to help assure your success. (only $6.95)

TWELVE SIMPLE LITTLE-KNOWN WAYS TO MAKE BIG MONEY FAST!

11. HOW TO SIT BACK AND RAKE IN A BUNDLE SELLING BOOKS BY MAIL:
Practically all mail order experts agree that absolutely nothing sells better by mail than books . . . and there's nothing that sells easier than books. Better yet — you stand to make bigger and faster net profits from selling books by mail than you could realize on any other items. You will be shown everything from A to Z. (only $6.95)

HOW TO SIT BACK AND RAKE IN A BUNDLE SELLING BOOKS BY MAIL



12. HOW TO WIN BIG CASH AND VALUABLE PRIZES CONTESTS:
This unique book quickly shows you all the important inside tricks. Opportunity to win national and local contests again and again. Cash, cars, homes, appliances, furs and vacations . . . No other publication on the market exactly like it. (only $6.95)

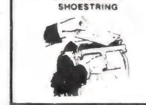
HOW TO WIN BIG CASH AND VALUABLE PRIZES CONTESTS

13. BIG FAST FULL TIME AND PART TIME PROFITS FOR WOMEN:
This book is a remarkable treasury of unique but common sense, easy to operate Little "big" money making businesses for many millions of today's serious and enterprising women interested in fabulous earnings, independence and security. Little or no investment and fast starting full- and part-time income increasing activities. (only $6.95)

BIG FAST FULL TIME AND PART TIME PROFITS FOR WOMEN

14. EASIEST AND FASTEST WAY TO START A SUCCESSFUL MAIL ORDER BUSINESS ON A SHOESTRING:
Simple, and most effective, step-by-step mail order start-up and operating instructions written especially for beginners. Crammed with vital facts . . . Covers every aspect of this exciting big money field. (only $6.95)

EASIEST AND FASTEST WAY TO START A SUCCESSFUL MAIL ORDER BUSINESS ON A SHOESTRING



15. HOW TO SEE THE WORLD . . . TRAVEL AND GET PAID WELL FOR IT:
Everyone enjoys traveling. But most people cannot afford to travel to those far away places they dreamed of visiting. Here's your chance to take in the wonderful sights throughout the world — and actually get paid for doing it. Yes, it's truly possible that this little known strictly legal method could provide you with the information for doing it. (only $6.95)

HOW TO SEE THE WORLD ... TRAVEL AND GET PAID WELL FOR IT

Fifteen Ways For You To Have Bulging Bank Accounts, Beautiful Homes, Expensive Clothes, Jewelry, Exotic Vacations . . . The Very Best Colleges For Your Kids . . . Plus, Keep A Steady Income Flowing In!

Everyone of the 15 Manuals' home-based businesses, can be successfully operated, by a single person, retirees, unemployed people — most ideal for husband/wife teams — and can be, almost, instantly turned into an enterprising family operated business, kids can help too. With everyone pitching in . . . your business could suddenly take off, and profits could increase fast!

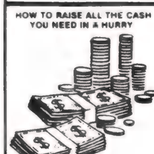
More Businesses You Operate . . . More Money You Make . . . Guarantees You Riches Beyond Your Wildest Dreams!

So, be sure to keep in mind that: Even though, it's true — some good money could be made with just a single one of these start-up success manuals working for you . . . but, much better than that, you could give yourself a greater opportunity to make your profits multiply much faster, by simply putting together a super powerful profitable combination of five, ten, or more of these fifteen — fastest, easiest . . . proven profitable businesses out of over a thousand in our files. Imagine having them all operating, and bringing in big hefty profits for you, at the same time! But you must send your order in right away. Supplies are extemely limited at these special introductory low prices!

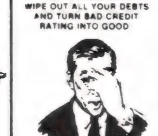
FREE The More Success Manuals You Order The More FREE Limited Editions You Get

Buy Any 2 to 5 SUCCESS MANUALS And Get Free! Any *One* Of The Three LIMITED EDITIONS Below, or Buy Any 6 to 10 SUCCESS MANUALS And Get Free! Any *Two* Of The Three Below, or Buy Any 11 to 15 SUCCESS MANUALS And Get Free! All Three Below.

HOW TO RAISE ALL THE CASH YOU NEED IN A HURRY



HOW TO QUICKLY WIPE OUT ALL YOUR DEBTS AND TURN BAD CREDIT RATING INTO GOOD



HOW TO RETIRE YOUNG AND LIVE LUXURIOUSLY ON VERY LITTLE MONEY

HOW TO RAISE ALL THE CASH YOU NEED IN A HURRY.
If you need $5,000, $20,000, $100,000 or more to help get your new business off the ground — then, you'll most certainly want to read this book. Even if you've already been turned down by banks . . . and finance companies.

HOW TO QUICKLY WIPEOUT ALL YOUR DEBTS AND TURN BAD CREDIT RATING INTO GOOD!
Now with this amazing book — you can stop bill collectors cold in their tracks. Here's your golden opportunity to get out of debt without borrowing.

HOW TO RETIRE YOUNG AND LIVE LUXURIOUSLY ON VERY LITTLE MONEY.
Finally — It's possible for you to say "goodbye" and "good riddance" to that old out-dated idea that you must remain on a boring nickel and dime wage slave, time-clock punching job until you reach 65.

FINALLY—A REAL OPPORTUNITY TO ENJOY A RICHER . . . BETTER LIFE

Our organization — Successful Business Publishers, offers what is perhaps the largest Collection of Unique 'Home-business' Beginner's Success Manuals in the world! Out of over a thousand of businesses in our organization's files — our home business specialists, have painstakingly selected The 15 Fastest, Easiest . . . Most highly Profitable part-time and fulltime Businesses you and other beginners can quickly start and easily operate from home — with very low or, virtually, no investment.

How many of these Proven, Highly Profitable Enterprises, can you combine together and successfully operate — and Benefit From At The Same Time? Possibly, all fifteen if you're that ambitious . . . The big Shrewd Corporations, call this almost secret method 'diversification' which is, merely, a high sounding big word, which simply boils down to — having a lot of different businesses, operating . . . and pouring fast-fat profits into your pockets, at the same time.

Let's face it — Plain common sense says that — the more of these 15 Proven Profitable Businesses, you choose to operate at the same time . . . the bigger, and faster your profits could be! Of course, our organization will profit a few more dollars if you choose to put five, ten or all fifteen of our proven profitable, start-up success manuals, to work for you.

But then — why should our making a modest few dollars of profit bother you — when it's you, who makes . . . and keeps all the income — no matter how much — Your different home businesses bring in? The demand for our unique wealth-building, Beginners Start-Up Success Manuals has been so overwhelming . . . and, understandably so too, since, there's absolutely nothing like them on the market!

Our No-Risk Success Guarantee To You

You must be absolutely, positively, and totally convinced that the actual money making success profitability of each manual is real — and may quickly increase your income or you may return everything within 10 days for a prompt no hassle refund.

Remember, those who snooze will certainly lose. However, those who choose right now, to begin — can win, and right now, while there's still time, is the best time to begin. You'll be making a very wise and highly profitable move. ORDER NOW!

Beginners Start-Up Success Manuals Order Form

Circle the manuals you are ordering
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15

I have circled above the catalog number of each Success Manual I am ordering, and I've included the proper amount to help cover shipping and handling, as indicated below. Also, I'm fully protected by your organization's strong, no-risk success guarantee that - unless I am totally convinced that the actual money making success profitability of my manual(s) is real - and may quickly increase my income. Also, I may return everything within 10 days, for a prompt, no-hassle, full refund.

Total Success Manuals Ordered ______________
Full Amount Enclosed ______________
Be sure to include proper shipping and handling fee - see charges below:

SHIPPING AND HANDLING CHARGES
Ordering just one Success Manual Add $1.25 for S&H
Ordering from 2 to 5 Success Manuals Add 90¢ per each manual
Ordering from 6 to 14 Success Manuals Add 50¢ per each manual
ENJOY BIG SAVINGS ON ORDERS FOR ALL 15 SUCCESS MANUALS - We pay all Shipping & Handling Cost. (a hefty savings of $7.50!).
Note: We pay shipping and handling on each Limited Edition Manual your order qualifies for.

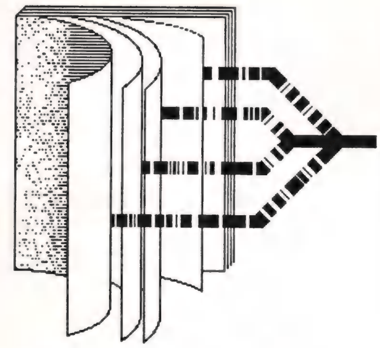
Check the box below which indicates each desired FREE Limited Edition Manuals' Title(s) which your order qualifies you to receive FREE:
- ☐ How to quickly wipe out all your debts and turn bad credit rating into good
- ☐ How to raise all the cash you need in a hurry
- ☐ How to retire young and live luxuriously on very little money

METHOD OF PAYMENT (all prices are in U.S. funds):
☐ My check or money order is enclosed (do not send currency through the mail).
Sorry - due to high percentage of sale charged by card companies - charge card orders not accepted.

SHIP TO
Name ______________
Address ______________
City ______________
State ______________ Zip ______________

Complete this order form and mail to:
SUCCESS BUSINESS PUBLISHERS
110 W. 5th Street
Winston-Salem, N.C. 27101

DDJ ON LINE



#: 8534 S0/General/DDJ office
05-Dec-86 17:31:07
Sb: Screen Weirdness!
Fm: Chris Johnston
71505,1752
To: All

I know that this is going to sound crazy, but I was sitting at my desk at work, talking to a coworker who was running Microsoft Word on a 6-MHz IBM PC/AT with a CGA. I was finishing my lunch, crunching on one of those big hard pretzels, when I noticed that every time I crunched the screen image moved! Nothing else did, just the image on the screen, which seemed to get smaller along the vertical axis, as if the tube lost power momentarily. This is not a real change in the image because you have to be crunching to see the effect, but everybody else who tried it saw the same thing. Our assumption is that the vibrations set up when you crunch are some multiple of the vertical frame rate, causing an apparent change in the screen. On some machines I can see the vertical retrace line when I try this. It doesn't seem to happen on a TV set.

Is anybody interested in trying this out and seeing if it is a universal phenomenon? We've been talking about trying to measure the 'crunch frequencies' with an accelerometer but haven't tried it yet.

P.S. It works with hard cookies, too.

***There is a reply: 8538

#: 8538
Sb: Screen Weirdness!
Fm: Ray Duncan [DDJFORUM]
76703,4265

It's not crazy. I've been noticing the same thing for about a year in my office. The funny thing is, I never see it on my *own* screen, but I can easily see it on the screen of the guy who sits on the other side of the room. I always assumed it was 'real' and wondered why he never said anything about it, and I never really could dream up a reason for it.

*** There is a reply: 8549

#: 8549
Sb: Screen Weirdness!
Fm: Pete Becker 76347,3151

I'd guess it's physiological. Maybe the vibration from the crunch shakes the retina or something. Such an effect would be more pronounced on distant objects: because they produce a smaller image, an identical displacement is proportionately much larger.

*** There is a reply: 8552

#: 8552
Sb: Screen Weirdness!
Fm: Chris Johnston
71505,1752

That makes a lot of sense. We have an HP spectrum analyzer and one of those accelerometer-hammer thingies that you use for measuring the vibration frequencies of mechanical parts. You give the item a whack to excite it, and then it senses the vibrations that result. We have been thinking about trying the pretzel crunch test with the accelerometer pressed firmly against a cheekbone to see what happens. My bet is the resultant frequencies are some multiple of the 60-Hz vertical frame rate.

*** There is a reply: 8554

#: 8554
Sb: Screen Weirdness!
Fm: Pete Becker 76347,3151

There probably aren't any well-defined resultant frequencies—just a bunch of garbage, which, of course, consists of all frequencies. I suppose a true controlled experiment would consist of putting the accelerometer against your cheekbone and hitting your jaw with a hammer so you have reproducible conditions. Any volunteers?

*** There are replies: 8555,
8603

#: 8603
Sb: Screen Weirdness!
Fm: Chris Johnston
71505,1752

No, I think I will skip the 'hit your jaw with a hammer part.' Somebody at work did suggest putting strain gauges on the pretzel! I'll try to remember to fire up the accelerometer/frequency analysis system and try out the less painful part of the test. I'll let you know!

#: 8555
Sb: Screen Weirdness!
Fm: jhon stanley 73765,1026

The wiggle is not psychological—it really exists. It is due to the vibration of the muscles of the eye, which try to correct themselves and therefore move the eye. It is so common that the brain automatically corrects for it. It can do a great job correcting those things that do not move but has trouble with moving things—like, for example, a band of light sweeping down the face of a CRT. The region of the CRT lit at any given time is small, maybe ½—1-inch tall. If your eye moves up while the band moves down, the screen will appear taller. Likewise, move the eye down and the picture gets smaller (somewhat like moving an original on the copy machine while the machine scans it). So, vibrate, and the picture wiggles.

I have shown this to many people, and they are always amazed. They don't believe it until they try it themselves.

*** There are replies: 8564,
8604

#: 8604
Sb: Screen Weirdness!
Fm: Chris Johnston

71505,1752
As I understand it, the eye wiggles a little all the time because nerves that are continuously stimulated shut down afer a little while. The places where the receptors see light and dark alternate is near an edge. At a low level, we have an automatic edge discriminator built into the system.

#: 8564
Sb: Screen Weirdness!
Fm: Pete Becker 76347,3151
To: jhon stanley 73765,1026
I said 'physiological,' not psychological! Your explanation sounds good.

#: 8637
Sb: Screen Weirdness!
Fm: jhon stanley 73765,1026
The same thing happens, I have noticed recently, with LED clocks. The digits are scanned, as in a CRT, so the numbers also wiggle when you chew.

*** There is a reply: 8639

#: 8639
Sb: Screen Weirdness!
Fm: Pete Becker 76347,3151
Synchronisity! I was at my sister's this afternoon, and she called me into the kitchen and asked me to click my teeth together while looking at the clock on her microwave oven. Of course, I wouldn't do anything so undignified, but I gave her your explanation.

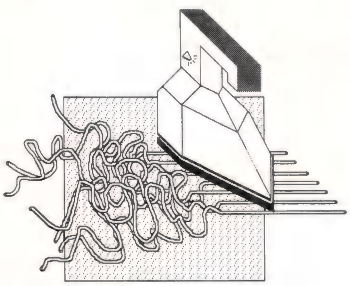
*** There is a reply: 8696

#: 8696
Sb: Screen Weirdness!
Fm: jhon stanley 73765,1026
I told a friend of mine at work to go brthththththphphphphtttt at the 19-inch monitor we use because it would do something neat. He gave me this funny look and a half-hearted brrththt. I told him to try again. He did, and the look on his face told me he saw the effect. Ain't science wonderful?

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 8.**

THE STATE OF BASIC



More BASIC Modules and Libraries

One of the shortcomings of the early versions of microcomputer BASIC was the absence of formal libraries of reusable code. The lack of support for multiline functions, callable BASIC subroutines, and local variables weakened any attempts to "simulate" BASIC libraries. The implementors of "new wave" BASICs, a phrase you will hear often, have recognized the need for supporting user-defined libraries. The solutions given by these new BASICs vary in the degree of flexibility and syntax, however. Let's take a look.

QuickBASIC lets you create libraries in two steps. First, you write the source code for a library and compile it into .OBJ form. The next step is to "build" an .EXE library file by using BUILDLIB.EXE, which is supplied with QuickBASIC. BUILDLIB.EXE is able to take one or more library object files and create a bigger library. To avoid confusion between libraries in .OBJ and .EXE files, you can rename the former as sublibraries. QuickBASIC permits your BASIC programs to use one library only. The default library is USERLIB.EXE, but you can instruct BUILDLIB.EXE to create libraries with other names. You must specify these library names when invoking QuickBASIC from the DOS command line.

The preceding discussion gives the impression that QuickBASIC supports one library at a time. The good news is that you can expand and update your .EXE library files by including new or modified sublibraries. To do this, you must store the .OBJ sublibrary files because you may rebuild a library periodically.

Example 1, below, shows a short subroutine that calculates the square root of a number using Newton's method. The library body has no formal declaration. Functions in QuickBASIC libraries are local to the library, so I have used a callable subroutine instead. A client QuickBASIC program need not make any special declarations to use the library subroutines; the burden falls on the program author to document external subroutine calls. The scheme of calling library subroutines in QuickBASIC offers a good degree of language extension.

In True BASIC, library files begin with the keyword *EXTERNAL*. Unlike QuickBASIC, all the subroutines and functions are accessible to the client program. Local variables are not automatically shared with other library routines or client programs; data flows through argument lists and data files. Unlike QuickBASIC, True BASIC enables your application program to use multiple libraries. You use the syntax *LIBRARY <library names list>* to indicate the files containing the sought libraries. Functions imported from libraries must be declared in *DECLARE DEF <function names list>* statements. This feature of True BASIC permits you to maintain libraries in a more independent way than you can in QuickBASIC. In addition, libraries can call other libraries in True BASIC—a valuable feature for modular software development.

Example 2, below, shows the square root function in a True BASIC library. The library/module loading feature enables you to do away with explicit *LIBRARY* and *DECLARE DEF* statements related to the loaded libraries. In that respect, True BASIC also offers a vehicle for language extension.

BetterBASIC supports modules, so much so that the implementation itself is modular. A customizable configuration text file is used to list the modules that are loaded into memo-

```
SUB SQROOT(X,S) static

   IF X )= 0 THEN
     ACCR = 1.0E-8
     S = X / 2
     WHILE ABS(S * S - X) ) ACCR
        S = (X / S + S) / 2
     WEND

   ELSE
     S = -1 ' result for a negative argument
   END IF
END SUB
```

***Example 1:** QuickBASIC library subroutine to compute the square root using Newton's method*

```
EXTERNAL ! declaration needed to define a True BASIC library
DEF FNSQRT(x)

IF X )= 0 THEN
 LET Accr = 1.0E-8
 LET S = X / 2
 DO WHILE ABS(S * S - X) ) ACCR
  LET S = (X / S + S) / 2
 LOOP
 LET FNSQRT = S
ELSE
 LET FNSQRT = -1 ! result for a negative argument
END IF

END DEF
```

***Example 2:** True BASIC library function to compute the square root using Newton's method.*

NEW RELEASE—Version 2.01
NOW Faster Than Ever!

	Ackerman		Sieve		I/O	
Marshal Pascal	11.9	3.5K	4.8	2.4K	1.8	5.1K
IBM Pascal	12.4	34.7K	11.7	27K	2.5	24.5K
Turbo Pascal	22.7	11.6K	14.2	11.5K	2.2	12.5K
Microsoft C 4.0	15.9	9.3K	5.8	6.5K	1.9	8.9K
Oregon Pascal-2	18.2	13.9K	7.2	11.7K	2.5	22K
	Sec.	Code Size	Sec.	Code Size	Min.	Code Size

POWER CORRUPTS!

Actually, it only took him a couple of days with Marshal Pascal. Marshal Pascal produces code smaller and faster than any other High Order Language on the PC. Period. Even optimized C compilers don't come near our performance. And Marshal Pascal allows you to generate an assembly language listing of your code, to boot. So it's not surprising that some people would fudge the truth.

However, speed and compactness are not enough. Power and ease of use are important, too. With Marshal Pascal you may address as much memory as your operating system allows and a variety of memory models are supported. Among the useful extensions incuded are: separate compilation of modules (both Modula-2 and Pascal forms), structured constants and structured function values, variable-length string types, overlays, long integers and procedural parameters. Our relocatable linker allows you access to the Microsoft family of languages and assemblers. The data-flow analyzer in the compiler automatically crunches your code far better (and with less headaches) than one can achieve by manually utilizing register variables. We support the Intel 8087-80287 math processors *inline*. If you don't have the math chip, then '87/287 simulation is a provided option. We also provide a number of compile options, including an optimization by-pass for speedier compiles, I/O "fine-tuning," and constant folds. A wealth of compile-time checks will help reduce your debug time enormously.

If that weren't enough, we also give you a **Turbo Translator,** allowing you to use the hundreds of application and utility programs written in Turbo Pascal® that are available in source from the public domain. Marshal Pascal supports most of the same extensions that are in Turbo Pascal, including low level hooks and inline coding, providing the Turbo user with the easiest and most comprehensive "step up" possible.

Space does not permit us to list all the features of Marshal Pascal, all of which are provided USER options, not additional COST options. Our compiler is not broken apart to make many products out of what should be only one good one.

Only, please folks, no matter how tempting it may be, don't fib to people once you start programming with Marshal Pascal!

the Price?

$189.

Marshal Pascal™

Supports PC-DOS®, **MS-DOS®**, **CP/M-86®**, **Concurrent Dos®**
and *soon* Xenix 286/386®!

To order your copy of Marshal Pascal call: (800) 826-2222
In California: (415) 947-1000

Marshal Language Systems

1136 Saranap Ave., Ste. P, POB 2010, Walnut Creek, CA 94595
Circle **no. 317** on reader service card.

(continued from page 138)

ry to provide your BASIC applications with additional routines. Some of the libraries are used to make Better BASIC compatible with BASICA.

To create a module in BetterBASIC, you create a new workspace in which you define local and exported routines as well as module initialization. You use a *PUBLIC* declaration as an export list; any routine not listed is strictly local. Creating a module in BetterBASIC is an interactive process. It involves a *MAKE MODULE <name>* command in which BetterBASIC requests you to verify your *PUBLIC* declaration and *MAIN* code (used to initialize the module). An affirmative answer puts your module in memory and makes its functions accessible as an extension of the language. Information is passed to module routines via argument lists, data files, or the use of pointers.

Example 3, below, shows a module that exports the BetterBASIC version of my square root function. Notice that BetterBASIC requires line numbers in some portions of the code. The declarations of variables are similar to those of Pascal (more about this in a future column). BetterBASIC uses a reserved identifier *RESULT* instead of the function name to return the result of a function. Also notice that the function arguments are not listed immediately after the function name but on the line that follows the function name declaration.

The implementation of libraries in the new wave BASICs, among other new features, offers an enhanced level of software engineering. The presence of software libraries acknowledges the following:

- the need for reliable software building blocks
- the shortening of development time by reusing existing routines
- support for structured and more systematic program development

Vote for your favorite feature/article. Circle Reader Service **No. 9**.

Vendors

BetterBASIC
Summit Software
106 Access Rd.
Norwood, MA 02062
(617) 769-7966
$199
Reader Service Number 50

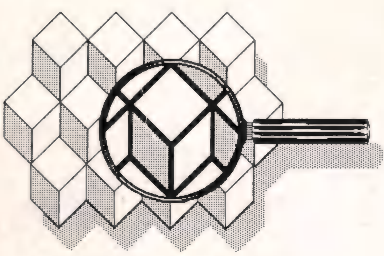
QUICKBASIC
Microsoft Corp.
16011
N.E. 36th Way
Redmond, WA 98052
(206) 882-8088
$99
Reader Service Number 51

True BASIC
True BASIC Inc.
39 S. Main St.
Hanover, HN 03755
(603) 643-3882
(call for prices)
Reader Service Number 52

```
PUBLIC: SQROOT

REAL FUNCTION: SQROOT
  REAL ARG: X
  REAL: S,Accr
    10 Accr = 1.0E-8
    20 S = X / 2
    30 WHILE ABS(S * S - X) > Accr DO
    40 S = (X / S + S) / 2
    50 REPEAT
    60 RESULT = S
END FUNCTION
```

Example 3: *BetterBASIC library function to compute the square root using Newton's method*

OF INTEREST



Languages

Pecan Software Systems' implementation of UCSD Pascal for the Apple IIGS is source-compatible with Apple Pascal. Extensions beyond Apple Pascal are offered in areas of multitasking, dynamic memory management, extended-precision arithmetic, and separate compilation. The software is ProDOS compatible and utilizes the new features of the Apple IIGS, including extended memory, sound, and graphics. The Power System Professional Pak is available for $199.95. Reader Service No. 16.
Pecan Software Systems Inc.
1410 39th St.
Brooklyn, NY 11218
(718) 851-3100

Hard Disks/Utilities

Storage Dimensions has introduced the AT160F, a 320-megabyte, high-performance, internal, dual hard-disk drive for IBM PC/ATs and compatibles. The AT160F reduces access time, breaks the 32-megabyte DOS barrier, and surpasses ROM BIOS maximum storage restrictions. It is fully compatible with the IBM PC/AT and its standard Western Digital controller card. Price for the AT160F ranges from $5,995 to $9,995, depending on storage capacity, number of drives, and the inclusion of controller. Reader Service No. 17.
Storage Dimensions
14127 Capri Dr.
Los Gatos, CA 95030
(408) 370-3304

The Storage Products Division of **Fujitsu America** has announced a high-performance, 5¼-inch, optical disk drive with a 600-megabyte formatted capacity. The M2505A WORM (write once, read many) drive utilizes a two-laser head design for fast throughput and interlink tracking to improve data reliability. The drive has a positioning time of 100 milliseconds, a rotational speed of 1,800 RPM, a transfer rate of 124K per second, and utilizes the Enhanced Storage Device Interface. The M2505A with M1080A controller is priced at $3,500, and media for the 5¼-inch drive is priced at $100. Reader Service No. 18.
Fujitsu America
3055 Orchard Dr.
San Jose, CA 95134
(408) 946-8777

Design Software has released DSBACKUP+ and DSRECOVER, hard-disk backup and protection utilities for the IBM PC, PC/AT, PC/XT, and compatibles. DSBACKUP+'s features include five-minute backup of a 10-megabyte hard disk, verification of data while backing up or restoring, data compression for up to 40 percent more data on each disk or cartridge, multiple volumes to allow backup and restore from more than one drive at a time, and the ability to back up only those files that have been changed since the last backup. DSRECOVER's features include undeletes in one step, views of all deleted files, and the ability to reconstruct original formatting when a hard disk is reformatted. DSBACKUP+ is priced at $79.95, and DSRECOVER is priced at $49.95. Reader Service No. 19.
Design Software
1275 W. Roosevelt Rd.
West Chicago, IL 60185
(800) 231-3088
In IL, HI, AK (312) 231-4540

SunDog Software Corp. has announced Squish, a 40K resident file-compression program that compresses files on both hard and floppy disks. When the files are used, expansion of data takes place spontaneously in memory rather than on disk, and no advance planning is needed to use the files. All programs that use standard DOS functions for reading and writing can use "squished" files. Squish is also compatible with other resident programs. The program is available for IBM PCs, PC/XTs, PC/ATs, and compatibles using DOS 2.0 and costs $79. Reader Service No. 20.
SunDog Software Corp.
264 Court St.
Brooklyn, NY 11231
(718) 855-9141

Fail-Safe from **CSSL** is a fault-tolerant system that allows IBM PCs and compatibles to continue operating even after catastrophic hard-disk failures. It is the first level of a multitiered system that comes in three configurations. The other configurations are DFT (Disk Fault Tolerant), a software and half-card version, and DFT II, a hardware-only version built around firmware and a controller card. Each configuration contains solutions to the most common problems found in personal computer system failures. Fail-Safe requires DOS 2.0 or later and 24K RAM. The single-unit PC version is available for $395. DFT, which is configured for a network linking up to 15 PCs, is available for $595. Reader Service No. 21.
CSSL Inc.
909 Electric Ave.
Seal Beach, CA 90740
(213) 493-2471

Rabbit Industries has introduced the MagicDrive, a quick and powerful hard-disk drive for Macintosh computers that have at least 512K RAM. It is available in 20-, 30-, 65-, and 235-megabyte versions and includes such features as automatic error detection and correction, daisy-chaining, automatic head parking, print spooling, password security, and backup utilities. Prices range from $699 to $3,399, depending on the version. Reader Service No. 22.
Rabbit Industries
4505 Spicewood Springs Rd., Ste. 304
Austin, TX 78759
(512) 343-0781

Tools

Flash-Up Windows from **The Software Bottling Company** is a memory-resident utility for creating, controlling, and managing menus and help windows for DOS, BASIC, Pascal, C, COBOL, FORTRAN, dBASE, and most other software. It lets programs control windows, allows you to define

Here's a plug for dBASE.™

Clipper™ NEWS RELEASE

FOR IMMEDIATE RELEASE

For information contact:
Nantucket
Nantucket Corporation®
12555 Jefferson Blvd.
Los Angeles, CA 90066
(213) 390-7923

CLIPPER™ NETWORKS DBASE APPLICATIONS

LOS ANGELES, California. . .Nantucket's Clipper now lets developers and business persons plug an unlimited number of workstations together to run their dBASE III and dBASE III PLUS applications, using Clipper's new networking capabilities.

This new release compiles programs to run on networks that support DOS 3.1 calls for networking functions, plus single-user programs for DOS 2.0 or greater.

Compiled Applications can be distributed freely, need no runtime module, no licensing fee or royalty. And there is no extra cost per user, regardless of how many users are connected to a Clipper network. Plus the new release now packs even more of Clipper's famous speed, on both single-user and networking applications.

The new Clipper also sports Expanded Memory support, additional functions and improved memo fields. The new release, dubbed Autumn '86, is not copy protected.

Clipper Autumn '86 is available for a suggested retail price of $695. Registered users of Clipper may upgrade to the new version for $139.

Circle **no. 220** on reader service card.

OF INTEREST
(continued from page 142)

windows, assigns windows to keys, and acts as a window macro enhancer by letting you send commands to running programs. Flash-Up Windows sells for $90. Reader Service No. 23.

The Software Bottling Company
6600 Long Island Expwy.
Maspeth, NY 11378
(718) 458-3700

A general-purpose set of development tools and C function libraries called Real-Tools is available from **Pioneering Controls Technologies**. Real-Tools comprises a screen-management system, windowing capabilities, user-defined graphics, and assorted utilities and library functions. It is priced at $99 for binary, $299 for library source, and $399 for complete source. Reader Service No. 24.

Pioneering Controls Technologies Inc.
510 Bering Dr., Ste. 300
Houston, TX 77057
(713) 266-8649

Csharp PC Drivers Package is a library of C language support routines for data acquisition and control hardware on the IBM PC, PC/AT, and compatibles from **Systems Guild**. It includes support for the Metrabyte Dash8 and Dash16 analog I/O boards, the Data Translation DT2801 and DT2808 analog I/O output boards, and the IBM PC DMA controller. Csharp PC Drivers Package can be used with the following C compilers: Microsoft 3.0 and 4.0, Lattice 2.15 and 3.10, and C86 from Computer Innovations. A special version of the product is available for use with Rational Systems' Instant-C incremental compiler. A source license for the Csharp PC Drivers Package costs $195. Reader Service No. 25.

Systems Guild
P.O. Box 1085
Kendall Square Station
Cambridge, MA 02142
(617) 451-8479

O88 is an optimizer compatible with the C Ware Corp.'s DeSmet C88 compiler. The product, introduced by **Key Software Products**, can run stand-alone or installed as an automatic part of the compilation pro-

cess. In minimal 8088 mode, O88 typically eliminates 4–13 percent of the instructions and simplifies 7–12 percent of those that remain. IBM PC/AT or compatible users can use 80188 mode to eliminate another 5–9 percent of the instructions. Programs that make heavy use of an 8087 or 80287 floating-point chip can use 8087 mode to achieve significant performance improvements. O88 is available for $49. Reader Service No. 26.
Key Software Products
440 Ninth Ave.
Menlo Park, CA 94025
(415) 364-9847

Six new programming toolkits for use with Kyan Pascal are available from **Kyan Software**. The toolkits save programmers time and help them add state-of-the-art graphics and other features to their Kyan Pascal programs. The toolkits run on Apple IIs with Kyan Pascal and are priced from $29.95 to $149.95. Reader Service No. 27.
Kyan Software
1850 Union St., #183
San Francisco, CA 94123
(415) 626-2080

Greenleaf Software has released DataWindows, a windows and data entry library for C language programmers. DataWindows' features include overlaid windows with screen management, transaction-oriented data entry, and more than 135 functions. Portions of the program's object code can be used in other programs without royalty obligations. DataWindows sells for $225, and the source code is available for an additional $225. Reader Service No. 28.
Greenleaf Software
1411 LeMay Dr., Ste. 101
Carrollton, TX 75007
(214) 631-0811

Cytek has released three new packages to enhance its Multi-C multitasking library. Multi-Comm is a full-featured communications library that supports high-speed, interrupt-driven data transfers, multiple device types in asynchronous or synchronous mode, and background communications by Multi-C tasks. Multi-Windows is a window development package for creating pop-up windows. Multi-Forms works with Multi-Windows to produce data entry and display screens. Source code is supplied for all hardware-dependent functions, allowing them to be used with any compiler-supported computer, including MS-DOS and ROM-based systems. Multi-Comm and Multi-Forms are priced at $149 each, and Multi-Windows sells for $295. Reader Service No. 29.
Cytek Inc.
805 Turnpike St., Unit 202
North Andover, MA 01845
(617) 687-8086

The C 386 Compiler and RLL 386 Relo-

(continued from page 149)

cation, Linkage and Library Tools package from **Intel Corp.** are designed to help speed development of both embedded and on-target 80386 application software. Both packages support all the 80386's features, capabilities, and operating modes. The compiler produces object code that is compatible with Intel's other 80386 languages. The RLL 386 tools package allows programmers to design protected multiuser and multitasking operating systems. The C 386 Compiler and RLL 386 package sell for $900 and $600, respectively. Reader Service No. 30.
Intel Corp.
Literature Dept. W338
3065 Bowers Ave.
Santa Clara, CA 95051
(408) 987-8080

Graphics

Microfield Graphics has introduced T8, a single-board graphics system for the IBM PC/AT, RT/PC, and desktop computers based on the Intel 80386. Based on a dual-processor architecture with 64-bit internal memory interface, the T8 is designed to meet the graphics requirements of high-end CAD/CAM, CAE, and mapping applications. Prices vary according to configuration. Reader Service No. 31.
Microfield Graphics Inc.
8285 S.W. Nimbus Ave., Ste. 161
Beaverton, OR 97005
(503) 626-9393

NSI Logic has introduced a half-size, enhanced graphics adapter called SMART (Single Monitor Adapter Technology) EGA. The adapter is compatible with any IBM PC software and operates in all the popular display modes on any standard EGA color monitor. It costs $549. Reader Service No. 32.
NSI Logic Inc.
Cedar Hill Business Park
257-B Cedar Hill Rd.
Marlboro, MA 01752
(617) 460-0717

Editor

UniPress Software has released a Unix-oriented text editor called vi-PLUS that has some features not found in standard vi, such as multiple windows, an interactive interface to Unix, and extensibility through macros. It is available for many computer systems running Unix, Xenix, Ultrix, and other Unix derivatives. The PC version sells for $645. Reader Service No 33.

UniPress has also released C-macs, a program editor for C programmers that is built on top of the company's Emacs editor. C-macs checks and balances parentheses and braces and permits programmers to define an indenting style. The editor can run make while an edit session is underway and maintains "tags" of all system components. The PC version of C-macs costs $645. Reader Service No 34.
UniPress Software Inc.
2025 Lincoln Hwy.
Edison, NJ 08817
(201) 985-8000

DDJ

ADVERTISER INDEX

*This advertiser prefers to be contacted directly: see ad for phone number.

ADVERTISING SALES OFFICES

Midwest
(317) 875-8093

Southeast
Gary George **(404) 897-1923**

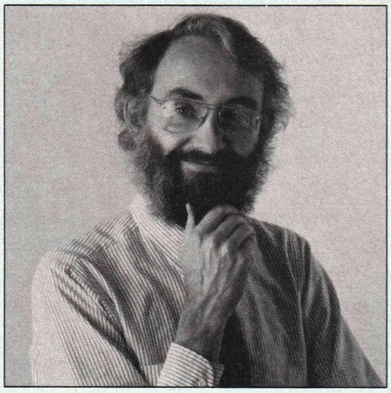
Northeast
Cynthia Zuck **(718) 499-9333**

Northern California/Northwest
Lisa Boudreau **(415) 366-3600**

Southern California/AZ/NM/TX
Michael Wiener **(415) 366-3600**

Advertising Director
Robert Horton **(415) 366-3600**

SWAINE'S FLAMES



A computer system that encapsulates the knowledge of experts (including heuristic decision processes) for retrieval by the inexpert but naturally intelligent is called an expert system. There are many expert systems in existence throughout the world today. When a British health association recently recommended that each health district in Great Britain avail itself of a (presumably human) expert on AIDS, it discovered that there were fewer AIDS experts than health districts in Britain. Enter the AIDS expert system.

At Warwick University in Great Britain, a team of computer scientists under the direction of Dr. Roger Brittain is scanning more than 100 articles a month on AIDS, building a knowledge base that will help doctors counsel AIDS patients and serve as a research and diagnostic tool. The project builds on a prototype system for rabies patients and is expected to cost some £20,000 and result in a piece of software that can run on a mainframe or microcomputer.

Despite some recent press grumblings about expert systems technology not being the miracle the same press had made it out to be, expert systems are useful tools in just such situations as AIDS diagnosis—in fact, many of the fundamental expert systems techniques were developed in a medical diagnostic framework called MYCIN. (We have received and expect to publish next month a MYCIN-like expert system.) The Warwick project sounds practical and may actually make a contribution to putting the brakes on the AIDS epidemic. That's great, and because I try to maintain a positive mental attitude in this column, I won't suggest that the three major American television networks are making a counterbalancing contribution to the spread of the disease with their refusal to carry condom advertising.

I learned about the AIDS expert system in a news item in the Christmas 1986 issue of a British weekly called *New Scientist*. This periodical is worth the time of any scientifically curious and naturally intelligent person on either side of the Atlantic. Erstwhile *DDJ* resident intern Dave Cortesi and I have shared a deep fondness for *New Scientist* for years (something like our fondness for Jon Bentley's Programming Pearls column in *IEEE Software* and our respect for at least the intentions of the best science fiction, this last being what Dave is currently writing). *New Scientist*'s snipes at the British government are often (to me) funny and its humor is largely (to me) incomprehensible, but everything else is gold. There is more to think about in six weeks of *New Scientist* than in twelve months of *Scientific American*.

A videophone system for the deaf that handles the bandwidth problem by abstracting essential expressive and gestural cues into an animated cartoon of the caller is something I've followed off and on for years; the latest word on this University of Essex project appeared in the October 23 issue of *New Scientist*. The November 27 issue talked about progress toward standardization on a Unix application interface, which could be the biggest boost for Unix since Bell Labs gave it to the universities. The December 18 issue had brief items on Texas Instruments' trenching technology for 3-D 4-megabit memory chips and on European research into sixth generation computers (see also "Sixth Generation Computers" by Richard Grigonis in the May 1984 *DDJ*). It's a good magazine.

January brought one of the most enjoyable of the computer trade shows: Macworld Expo. The atmosphere this year was particularly charged, and there were products and announcements significant enough to support the electricity.

Steve Jasik showed off his debugger MacNosy Part Two: The Debugger. His slogan: Beyond Discipline and Into Bondage. The Interface Builder from Expertelligence looked interesting: it lets you put a Mac interface on LISP programs. The Developers' Toolkit panel, with moderator Scott Knaster (manager of the Developer Technical Support Group at Apple), MicroPhone author Dennis Brothers, Jim Friedlander of Apple, and David Intersimoni of Borland, talked encouragingly about MacApp and APDA, the Apple Programmer's and Developer's Association.

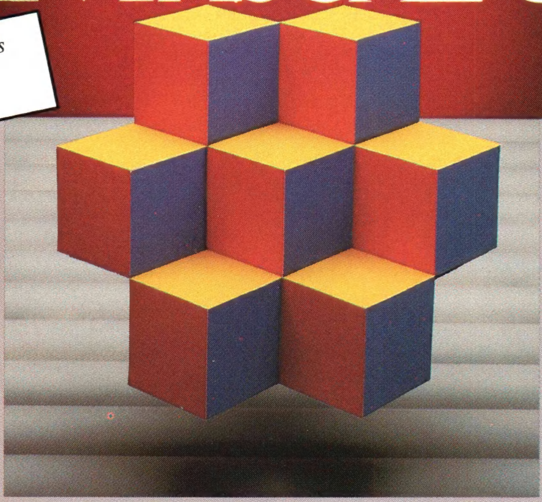
One East Coast writer out for the Expo was sometime *Rolling Stone* writer Steven Levy, who some think was the model for the John Travolta character in the less-than-perfect movie *Perfect*. Levy definitely was one of the founders of the Lunch Bunch, a group of technology writers who ate hamburgers together on two coasts. The Lunch Bunch has served up at least one book and has now spun off a dinner group gourmandizing in Silicon Valley under the label Nerd's Night Out. January's menu called for a discussion of Apple's new machines, but the most knowledgeable sources decided that they couldn't talk about that and canceled. How Apple has changed.

Michael Swaine

Michael Swaine
editor-in-chief

WHY LOGITECH MODULA-2 IS MORE POWERFUL THAN PASCAL OR C.

"A clear winner... The integrated editor is a joy to use." BYTE Magazine, Jan. '87



APPRENTICE PACKAGE $99

WIZARDS' PACKAGE $199

- Separate Compilation w/inter-module typechecking
- Native Code Generation
- Large Memory Model Support
- Most Powerful Runtime Debugger
- Comprehensive Module Library
- Maintainability
- Translator from Turbo and ANSI Pascal

WIN A FREE TRIP TO Switzerland

HOMELAND OF MODULA-2

Return your Modula-2 Registration Card or a reasonable facsimile* postmarked between March 1, 1987 and May 31, 1987 to be included in a once-only drawing!

Grand Prize: One week excursion for 2 in Zurich, Switzerland including a guided tour of ETH, the University where Modula-2 was created by Niklaus Wirth. European customers may substitute a trip to Silicon Valley, California.

Second and Third Prizes: LOGITECH C7 Mouse or LOGITECH Bus Mouse with Paint & Draw software—a $219 value, absolutely free!

*Write to Logitech, Inc. for a registration card facsimile.

NEW!

APPRENTICE PACKAGE $99

Everything you need to begin producing reliable maintainable Modula-2 code. Includes the Compiler with 8087 support, integrated Editor, Linker, and BCD Module. We're also including FREE our Turbo Pascal to Modula-2 Translator!

NEW!

WIZARDS' PACKAGE $199

This package contains our Plus Compiler—for professional programmers or for those who just want the best. The Plus Compiler with Integrated Editor requires 512K and takes advantage of the larger memory to increase compilation speed by 50%. Our Turbo Pascal to Modula-2 Translator is also included at no extra charge.

NEW!

MAGIC TOOLKIT $99

We've put our most powerful development tools into one amazing Toolkit for use with either the Apprentice or Wizards' packages. Highlighted by our Runtime Debugger, the finest debugging tool available anywhere, the Toolkit also includes our Post Mortem Debugger, Disassembler, Cross Reference utility and Version which keeps track of different versions of one program. Our MAKE Utility figures out module dependencies and automatically selects those affected by code changes to minimize recompilation and relinking. We also provide source code of our major library modules for you to customize—or just play with.

WINDOW PACKAGE $49

Now you can build true windowing into your Modula-2 code. Features virtual screens, color support, overlapping windows and a variety of borders.

ROM PACKAGE AND CROSS RUN TIME DEBUGGER $299

For those who want to produce rommable code. You can even debug code running in ROM from your PC.

Turbo Pascal is a registered trademark of Borland International.

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

To place an order call toll-free:

800-231-7717

In California:

800-552-8885

YES! I want the spellbinding power of LOGITECH Modula-2!

- ☐ Apprentice Package **$99**
- ☐ Wizards' Package **$199**
- ☐ Magic Toolkit **$99**
- ☐ Window Package **$49**
- ☐ ROM Pkg/Cross RTD **$299**

Add $6.50 for shipping and handling. Calif. residents add applicable sales tax. Prices valid in U.S. only.

Total Enclosed $________

☐ VISA ☐ MasterCard ☐ Check Enclosed

Card Number Expiration Date

Signature

Name

Address

City State

Zip Phone

LOGITECH

LOGITECH, Inc.
805 Veterans Blvd. Redwood City, CA 94063
Tel: 415-365-9852

In Europe:
LOGITECH SA, Switzerland
Tel: 41-21-879656 • Telex 458 217 Tech Ch

In Italy:
Tel: 39-2-215-5622

Circle **no. 257** on reader service card.

NEW **Turbo C**®

Turbo C: The fastest, most efficient and easy-to-use C compiler at any price

Compilation speed is more than 7000 lines a minute, which makes anything less than Turbo C an exercise in slow motion. Expect what only Borland delivers: Quality, Speed, Power and Price.

Turbo C: The C compiler for amateurs and professionals

If you're just beginning and you've "kinda wanted to learn C," now's your chance to do it the easy way. Like Turbo Pascal, Turbo C's got everything to get you going.

If you're already programming in C, switching to Turbo C will considerably increase your productivity and help make your programs both smaller and faster. Actually, writing in Turbo C is a highly productive and effective method—and we speak from experience. Eureka: The Solver™ and our new generation of software have been developed using Turbo C.

Turbo C: a complete interactive development environment

Free MicroCalc spreadsheet with source code

Like Turbo Pascal® and Turbo Prolog,™ Turbo C comes with an interactive editor that will show you syntax errors right in your source code. Developing, debugging, and running a Turbo C program is a snap.

Turbo C: The C compiler everybody's been waiting for. Everybody but the competition

Borland's "Quality, Speed, Power and Price" commitment isn't idle corporate chatter. The $99.95 price tag on Turbo C isn't a "typo," it's real. So if you'd like to learn C in a hurry, pick up the phone. If you're already using C, switch to Turbo C and see the difference for yourself.

System requirements

IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. One floppy drive. 320K.

**Introductory price—good through July 1, 1987*

Technical Specifications

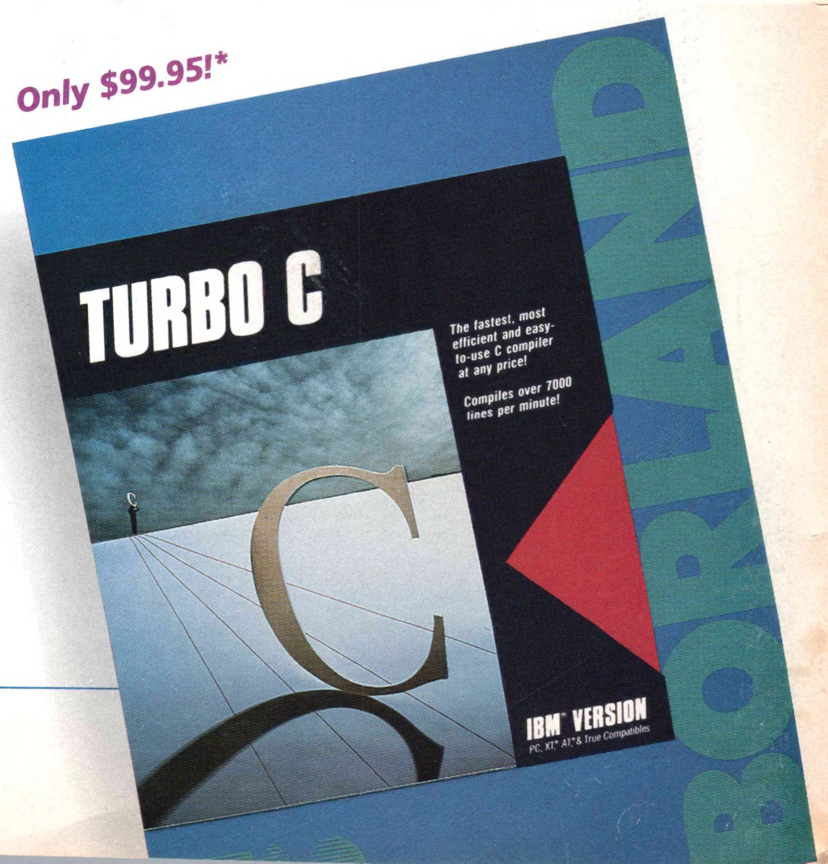
- ☑ *Compiler: One-pass compiler generating linkable object modules and inline assembler. Included is Borland's high performance "Turbo Linker." The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).*
- ☑ *Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.*
- ☑ *Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.*
- ☑ *Links with relocatable object modules created using Borland's Turbo Prolog into a single program.*
- ☑ *ANSI C compatible.*
- ☑ *Start-up routine source code included.*
- ☑ *Both command line and integrated environment versions included.*

Sieve benchmark (25 iterations)

	Turbo C	Microsoft® C	Lattice C
Compile time	**3.89**	16.37	13.90
Compile and link time	**9.94**	29.06	27.79
Execution time	**5.77**	9.51	13.79
Object code size	**274**	297	301
Price	**$99.95**	$450.00	$500.00

Benchmark run on a 6 Mhz IBM AT using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51; Lattice C version 3.1 and the MS object linker version 3.05.

Only $99.95!*



BORLAND
INTERNATIONAL

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Vive la différence

For the dealer nearest you or to order by phone call

(800)255-8008

in CA (800) 742-1133 in Canada (800) 237-1136

Circle **no. 161** on reader service card.